AFIT UAV SWARM MISSION PLANNING AND SIMULATION SYSTEM

THESIS

James N. Slear, Captain, USAF

AFIT/GCE/ENG/06-08

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCE/ENG/06-08

AFIT UAV SWARM MISSION PLANNING AND SIMULATION SYSTEM

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Engineering

James N. Slear, BS

Captain, USAF

June 2006

AFIT/GCE/ENG/06-08

AFIT UAV SWARM MISSION PLANNING AND SIMULATION SYSTEM

James N. Slear, BS
Captain, USAF

AFIT/GCE/ENG/06-08

Abstract

The purpose of this research is to design and implement a comprehensive mission planning system for swarms of autonomous aerial vehicles. The system integrates several problem domains including path planning, vehicle routing, and swarm behavior.

The developed system consists of a parallel, multi-objective evolutionary algorithm-based path planner, a genetic algorithm-based vehicle router, and a parallel UAV swarm simulator. Both the path planner and the UAV swarm simulator are developed on AFIT's Beowulf parallel computer clusters.

An extensive set of tests are performed to validate the system components as well as the system integration. Tests focus on two primary objectives: efficiency and effectiveness.

The simulator is interfaced with a visualization system that serves as both an iterative design tool and as a mission playback tool. As a design tool, the visualization system provides rapid feedback, allowing developers to quickly observe the effects of model changes on its behavior. As a mission playback tool, decision makers and mission planners can view mission scenarios played out with different sets of parameters.

Novel aspects of this research include: integrating terrain following technology into a swarm model as a means of detection avoidance, combining practical problems of path planning and routing into a comprehensive mission planning strategy, and the development of a swarm behavior model with path following capabilities.

The culmination of this effort is the development of an extensible developmental model for swarm behavior. Discussions on the of the system's capabilities and limitations are presented along with recommendation for further development.

Acknowledgments

I would like to express my sincere appreciation to Dr. Ken Melendez for all his efforts during the development of this work. Dr. Melendez was a full partner in the development of the UAV swarm simulator. His tireless efforts greatly contributed to this research.

Mr. John Zaloudek of Virginia Polytechnic Institute and State University was instrumental in the development of the parallel path planner. First, he served as a sounding board for the many ideas I had during this research. Later, he provided assistance as an emergency code debugger and analyst. He also developed the modified convex hull algorithm (see Chapter 3).

My Advisor, Dr. Gary Lamont, deserves much thanks for his patience during the past two years. His feedback always kept me from digging too big a hole for myself.

I'd also like to thank out technicians Dave Doak and Jason Speckman for their efforts in helping us to install and upgrade our software.

Lt. Nick Amato of AFRL deserves much credit for the visualization of the swarm simulator. Lt. Amato developed the interface between our model and the visualization program.

James N. Slear

Table of Contents

Page

Abstract iii

Acknowledgements iv

Table of Contents v

List of Figures vii

List of Tables x

I. Introduction 1-1

Problem Statement 1-1
Fundamental Concepts of Research 1-3
Research Goal, Objectives, and Approach 1-6
Sponsors 1-9
Thesis Overview 1-10

II. Background and Historical Perspective 2-1

Path Planning 2-1
The Vehicle Routing Problem 2-15
AFIT UAV Swarm Simulator 2-20
Summary 2-27

III. High Level Design 3-1

Design Objectives 3-1
System Level Design 3-24
Summary 3-31

IV. Low Level Design 4-1

Software Development Approach 4-1
Parallel Path Planner 4-4
Swarm Simulation 4-14
Summary 4-26

V. Design of Experiments 5-1

Validation of the Path Planner 5-1
Integration of the Path Planner with the Router 5-5
Evaluating the Improved Swarm Simulator 5-5
Summary 5-10

VI. Results 6-1

Validation of the Path Planner 6-1
Evaluating the Improved Swarm Simulator 6-6
Summary 6-16

VII. Conclusion and Future Results 7-1

Satisfaction of Objectives 7-1
Improvements to the Path Planner 7-1
Improvements to the Swarm Simulator 7-3
Future Capabilities 7-5

Appendix A. Modified Convex Hull Algorithm A-1

Appendix B. Basic MPI Constructs B-1

Bibliography R-1

List of Figures

Figure Page

1.1 Swarm Model Visualization 1-10

2.1. A Two DOF Arm in an Operational Space with Obstacle.............................. 2-3

2.2. Configuration Space... 2-3

2.3. Approximating B-Splines by Curve Sampling 2-12

2.4. An Illustration of Intervisibility.. 2-14

2.5. An Illustration of Hideability.. 2-15

2.6. Reynolds' Distributed Behavior Model 2-21

2.7. Reynolds' Behavior Hierarchy 2-24

2.8. Reynolds' Seek and Flee Steering Behaviors 2-25

2.9. Reynolds' Pursuit and Flee Steering Behaviors.. 2-26

2.10. Reynolds' Arrival Steering Behavior.. 2-26

3.1. Example of a Route Intersecting a Detection Zone 3-5

3.2. High-Level View of Path Planning Algorithm 3-7

3.3. Chromosome Structure of the Path Planner.. 3-8

3.4. Population Initialization of the Path Planner 3-9

3.5 Crossover Operator 3-11

3.6 Mutate Add Operator with Repair 3-12

3.7 Mutate Delete Operator with Repair.. 3-13

3.8. Mutate Alter Operator with Repair.. 3-14

3.9. Head-on View of Swarm Formation... 3-17

3.10. Swarm Separation 3-18

3.11. Cohesion Rule Applied to Swarm Member 3-20

3.12. Separation Rule Applied to Swarm Member 3-21

3.13. The Target Seeking Rule 3-21

3.14. Swarm Split Operation 3-23

3.15. High-Level Design of AFIT UAV Swarm Mission Planning System.......... 3-25

3.16. Interfacing the Planner and the Router…………………………………...3-27

3.17. Requesting Additional Routes from the Planner 3-29

3.18. Parallelization of the Path Planner 3-31

4.1. Class Diagram of the Path Planner 4-4

4.2. Chromosome Initialization.. 4-6

4.3. Data Members of the Path ADT. 4-8

4.4. Invoking the Path Planner from the Router 4-12

4.5. Decoding Routes into Triplets 4-13

4.6. Low-Level Parallelization of the Path Planner 4-14

4.7. SkyView Terrain Visualization.. 4-25

5.1. Terrain Avoidance Problem.. 5-2

5.2. Nellis Route over DTED.. 5-3

5.3. Nellis Route over Navigation Chart... 5-4

5.4. Top Down View of Swarm After Split Operation 5-7

6.1. Path Planner Navigating Simple Terrain 6-1

6.2. Route Optimized for Cost 6-2

6.3. Optimized Low-Risk Route 6-3

6.4 TFO Optimized Route.. 6-4

6.5 Pareto Front for Nellis Route 6-5

6.6 Experiment 1: Average Neighborhood Size 6-7

6.7 Experiment 1: Average Threshold Violation. 6-7

6.8 Experiment 1: Average Threshold Violation for Small Swarms....................... 6-8

6.9 Experiment 2: Neighborhood Deterioration 6-9

6.10 Experiment 2: Average Threshold Violation 6-9

6.11 Experiment 2: Threshold Violation for Small Swarms................................. 6-10

6.12 Experiment 2: Swarm Fan Out 6-10

6.13 Experiment 3: Neighborhood Deterioration 6-11

6.14 Experiment 3: Average Threshold Violation 6-11

6.15 Experiment 3: Threshold Violation for Small Swarms................................. 6-11

6.16 Experiment 4: Neighborhood Deterioration 6-12

6.17 Experiment 4: Average Threshold Violation... 6-13

6.18 Experiment 4: Threshold Violation for Small Swarms................................. 6-13

6.19 Experiment 5: The Effect of Synchronization Period on Model Stability..... 6-14

6.20 Parallel Runtimes as a Function of Problem Size and Number of Procs....... 6-15

A.1. Octagon Placement A-1

A.2. Placement of Octagonal Vertices on Midpoint. A-2

List of Tables

Table Page

3.1. Neighborhood Definition 3-19

4.1. Typical Swarm Simulation Event Sequence 4-24

5.1. Parallel Problem Test Cases 5-5

5.2. Swarm Model Behavior Enhancements 5-6

5.3. Parameters of Model Scalability Experiment 5-9

6.1. Fitness Function Component Scores for Nellis Route 6-3

6.2. Nellis Route Evaluations 6-5

6.3. Average Runtimes for Various Parallel Configurations 6-6

6.4. Runtimes for Parallel Problem Execution 6-15

AFIT UAV SWARM MISSION PLANNING AND PARALLEL SIMULATION SYSTEM

1. Introduction

1.1. Problem Statement

Path planning is the process of designing a sequence of states through which an object must assume in transiting from an initial state to a goal state [28]. Path planning optimization is a process that proscribes a particular plan for reaching a goal state from an initial state at a minimal cost. A path planning algorithm is a sequence of steps taken to calculate a path plan given knowledge of the path environment and a set of conditions or constraints that must be adhered to. Many successful path planning algorithms have been developed over the years [5] [8] [9] [22] [44] [59] [67] [68]. These algorithms vary in their effectiveness and efficiency based primarily on the specific formulation of the path planning problem and the number of variables and constraints required to solve the problem.

The Vehicle Routing Problem (VRP) is defined as the task of assigning a set of vehicles, each with a limited range and capacity, to a set of locations or targets that must be visited [62]. The VRP is an NP-complete problem [21]. Such problem classes do not lend themselves to deterministic problem solving methods because the runtime of these approaches grows exponentially with the problem size. Many stochastic methods have been used to provide "good" solutions to the VRP in reasonable time [49][62]. These stochastic methods achieve their results by generating feasible solutions and then improving these results through successive refinements using heuristics. In all of these methods, the set of inputs to the problem includes a cost associated with traveling from one target or location to another. These costs can be represented simply as the distance between the locations or they can contain other costs associated with the problem's constraints. The costs are usually stored in a table or matrix, and are referenced continuously by the algorithm to compute the overall cost of a particular solution.

Representing cost as a fixed input is adequate for routing problems in which distances between targets are large enough to ignore the added path lengths resulting from having to make series of turns in order to change heading from one location to another. However, when the target layout is such that the distances between the targets are as near as several turn radii of an aircraft apart, then the cost of traveling between any two targets must consider the heading at which the aircraft arrived at the initial location and the heading the aircraft must assume to vector itself towards the next target.

Taking this into account, algorithms that solve the VRP must calculate the cost of every assignment from scratch in order to accurately represent the cost associated with that assignment. In this research, a path planning algorithm is developed that simplifies cost calculations for the set of possible paths through a given set of targets by first considering target triplets, each consisting of a start node, midpoint, and end node. The algorithm calculates the optimal path through each triplet, and these triplet paths are concatenated with other triplets to quickly and accurately calculate the actual cost of a vehicle assignment. This information is tabularized and fed in as inputs to programs such as the Genetic Vehicle Router (GVR) [49] where good assignments can be made; but this time, the costs associated with these assignments is more representative. The goal is not merely to calculate the true cost of a particular assignment made by the GVR but to influence the GVR to make better assignments using the more complete cost information.

1.2 Fundamental Concepts of Research

This section describes the major research areas and concepts used in this research. First, a brief description of evolutionary algorithms and their application as a mission planning system are given. Next, the concept of Terrain Following is defined. The section concludes with a brief discussion on parallel discrete event simulation (PDES).

1.2.1 Evolutionary Algorithms. High dimension optimization problems can not be solved in a reasonable time by traditional, deterministic search algorithms. Their time complexity scales exponentially with the problem size thus prohibiting the employment of exhaustive search methods. Evolutionary algorithms (EAs) use biologically-inspired methods to create and evolve solutions in a reasonable amount of time [2] [38]. Populations of candidate solutions evolve through the use of genetic operators such as selection, recombination, and mutation. After multiple generations members of the population tend to converge to near-optimal solutions.

In optimization problems, EAs attempt to find the optimum (min or max) cost solution from a set of candidates. This cost can represent a single datum calculated by an evaluation function, or it can represent an aggregate of cost functions. When problems require minimization of multiple competing, cost elements, a trade-off is established between the set of competing requirements. In these instances, multi-objective evolutionary algorithms (MOEAs) can provide a decision maker with a variety of candidate solutions, each representing a level of optimization of one parameter with respect to another [11][18]. MOEAs produce a set of solutions where each solution represents an optimal cost in terms of one parameter for a corresponding cost associated with one or more others. Presenting decision makers with a set of choices allows them to see the trade-offs between competing objectives and make choices based on acceptable levels of these objectives.

In this research, an MOEA is developed for path planning where the objectives are cost, (encompassing distance traveled and the amount of climbing a vehicle does), and risk (resulting from flying through areas of threat). The solution set contains a selection of routes such that each route has the lowest cost associated with a particular level of risk and vice versa. The development of this MOEA is detailed in Chapters 3 and 4 while evolutionary algorithms are described from a historical perspective in Chapter 2.

1.2.2 Terrain Following. Terrain following (TF) is a mode of flight in which an aircraft maintains a fixed altitude above ground level (AGL) and flies low (on the order of a few hundred feet) through an area of interest. Naturally, this type of flying involves a great deal of climbing and descending, a costly operation. DoD terrain following missions have been carried out for many years dating back to Cold War missions flown by F-111 Raven aircraft [19]. More recently TF missions have been flown by U.S. Special Operations Command (USSOCOM) in using MC-130 gunships and a variety of rotary-wing platforms [55].

The idea behind TF is to remain hidden from enemy air defenses without using stealth technology. This technique of hiding within rugged terrain is known as terrain masking. Terrain masking (TM) algorithms determine a route of flight in which an aircraft can move toward a target or location of interest while remaining masked from enemy air defense radar by the surrounding terrain. Often routes calculated by TM algorithms have significant climbing and descending costs associated with them. The process of picking the best-masked routes with the least possible cost in terms of climbing and overall distance traveled is known as Terrain Following Optimization (TFO). Current terrain following Optimization algorithms developed by Air Force Research Laboratory (AFRL) use a set of deterministic operators that produce an optimal route selected from a limited subset of possible route choices. In this research, the multi-objective route optimization algorithm seeks to find better routes by exploring larger areas of the search space.

TF operations and TM technology can benefit the development of swarms of autonomous UAVs. First, swarm components need to be inexpensive and potentially expendable. Naturally, stealth technology in individual swarm components would be both costly and their loss could result in undesirable disclosure of technology to enemy forces. Secondly, even if stealth technology is not too costly for swarm elements, maintaining the stealth property of an entire swarm would require highly regulated flight patterns. Such patterns would inhibit the self-organizational exploratory nature of autonomous UAVs. By incorporating TM technology into swarm elements, swarms could gain the benefits of remaining hidden without the expense of stealth technology and the computational expense of maintaining a stealthy formation.

1.2.3 Simulation and Visualization. Routing algorithms and vehicle assignments represent a static view of a mission. Simulation and visualization of a mission enhances the commander's view of the battle plan and identifies some potential pitfalls. To this end, the routes and assignments generated from the given problem instances are visualized using the SPEEDES parallel discrete event simulator and the SkyView visualization tool [56][57]. Real world digital terrain data is used by the algorithms and shown in the visualizations along with realistic threat representation. Additional information about SPEEDES and SkyView are contained in Chapter 4.

Currently, AFIT maintains a UAV swarm simulation [13] [23] which models various swarm behaviors developed for self-organizing autonomous air vehicles. This research adds a routing capability to the model so that swarms can proceed to assigned targets while still exploring and exploiting the battle space for targets of opportunity discovered by their sensors. The initial routes and assignments used in the simulation are those generated from the problem instances in this research.

1.3 Research Goal, Objectives and Approach

The research goal is the improvement of routing and mission planning capabilities for UAVs and terrain following air vehicles. In this effort there are three main objectives:

1. Develop a multi-objective evolutionary algorithm for efficient path planning

2. Develop a parallel system that computes individual route segments for use in a GVR algorithm

3. Improve AFIT's parallel swarm simulator by incorporating path-following capabilities with existing swarm behavior and observe the extent to which limitations of fixed-wing aircraft constrain model behavior.

1.3.1 Objective 1: Path Planning Algorithm. The first objective concerns the development of a robust path planning algorithm for terrain following missions. Since all routes have both a cost and a risk associated with them, path planning is naturally expressed as a multi-objective minimization problem. Most often, decreasing the cost of the path, i.e. the path length and the amount of climbing required to navigate the terrain, results in increasing the risk associated with enemy air defenses. Likewise, a path generated to avoid intersection with all enemy air defense radar systems results in increased path cost. Single objective problem formulations for path planning often use constraints such as obstacle and threat avoidance and then calculate the least-cost path available that adheres to all constraints [53][68]. Other single objective problem formulations treat constraints as components of the solutions fitness [67]. Problems defined in this way have weights assigned to each objective and the resulting fitness is an aggregation of component scores. The common disadvantage of these approaches is two fold. First, a risk free path may not exist or its cost may exceed the capabilities of the aircraft. Second, paths containing an acceptable level of risk may have a substantially lower cost than a completely risk adverse path if one exists. A multi-objective objective approach provides a commander with a choice of routes with cost proportional to their level of risk. This empowers the commander to choose the acceptable level of risk and obtain the least-cost path associated with that choice. Due to the intractability of the path planning problem, an evolutionary approach is developed to produce low cost routes in a reasonable amount of time. The multi-objective evolutionary algorithm design for path planning is described in detail in Chapter 3.

Validation of the path planning component is accomplished through a set of experiments (see section 5.1) aimed at testing the planner's ability to produce feasible routes, to avoid terrain, and minimize exposure. Analysis of this data accompanies visualizations of solutions to various problem instances. (See Section 6.1).

1.3.2 Objective 2: Parallelization and integration into GVR algorithm. The Genetic Vehicle Routing algorithm [49] [61] uses an evolutionary approach to find an optimal assignment of vehicles to targets for combat or reconnaissance missions. The algorithm uses as its set of inputs, the cost associated with traveling between any two target locations. This cost reflects only the Euclidian distance between the targets. In order to include the cost incurred by turning from one location and proceeding to another, which increases the path length, the actual cost of traveling between two locations must include the direction from which the aircraft approached the first target and the direction the aircraft will depart the second target in route to a subsequent target. By calculating the cost associated with traveling from each location to every other through an intermediate point, the true cost of an entire target assignment can be calculated by concatenating the set of triplets to construct the route. The generation of optimal route triplets scales as $\boldsymbol{O}(n^3)$ compared to the $\boldsymbol{O}(n^2)$ cost of optimizing pair-wise links. This limits scalability but is less costly than the exponential alternative of enumerating and calculating all possible permutations of complete route assignments. To offset some of the cost of enumerating triplets, the path planning algorithm is parallelized, solving multiple triplets concurrently. The output data from the path planner is then given as input to the GVR algorithm which has been modified to use this new data in its evaluation function. The result is an optimal assignment of aircraft to targets based on the true costs of completing the routes.

Testing on this component focuses on the efficiency and scalability of the parallelization of the path planner and its ability to answer queries from the vehicle router.

1.3.3 Objective 3: *Parallel Swarm Simulation Model.* AFIT's swarm simulation model [13][23] models a swarm of autonomous air vehicles with a set of three behaviors. The first behavior is a tendency to remain together. The second behavior is a tendency to maintain a safe distance from one another. The third behavior is for the swarm members to align themselves together toward a particular direction. The swarm simulation is extended in this research to include a routing capability that guides the swarm along a route generated by the path planner and the GVR optimizer while still adhering to the three existing behaviors. Testing of this research component focuses on the ability of the swarm model to conform to established rules while adding new capabilities. The design of experiments in Section 5.2 provide metrics to measure swarm formation correctness and to determine the effects of additional functionality on the swarm's ability to adhere to the formation rules given in Section 3.5. While the swarm model includes many abstractions, it attempts to account for limitations in aircraft performance to the extent that they limit swarm behavior. Figure 1.1 gives a sample visualization of the swarm model developed in this research.

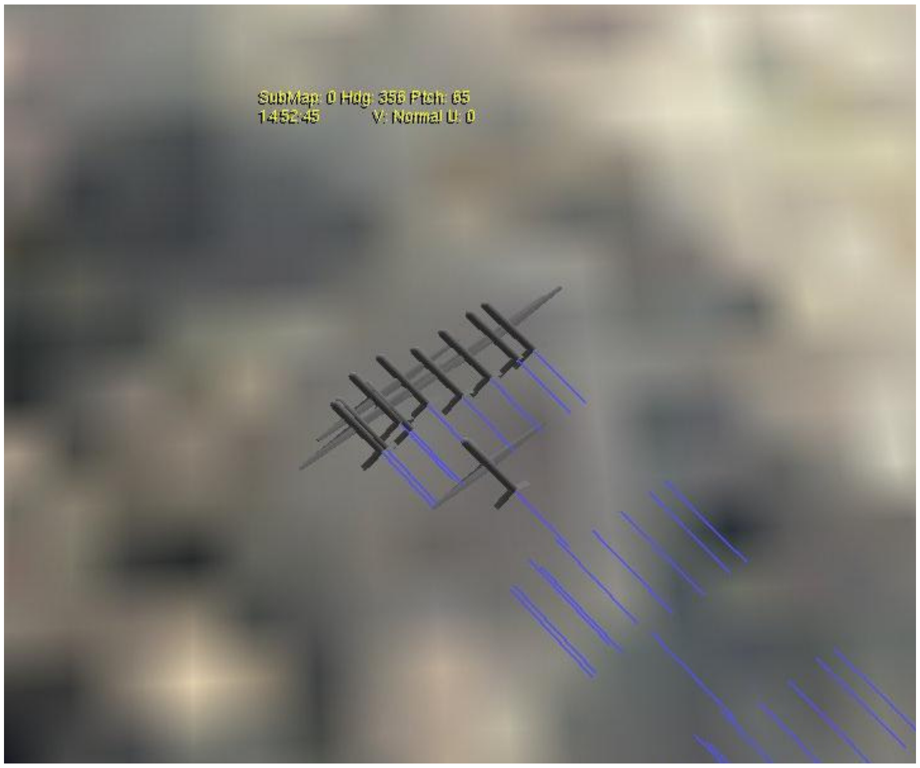


Figure 1.1 Swarm Model Visualization.

1.4 Sponsors

This research is sponsored by the Information Directorate, Air Force Research Laboratory (AFRL), Wright Patterson Air Force Base, Ohio. The mission of the Information Directorate is "the advancement and application of information systems, science, and technology to meet Air Force unique requirements for Information Dominance and its transition to air and space systems to meet war fighter needs." This research supports this mission by offering a capability to enhance mission planning and integration of autonomous vehicles into the mission planning process. The intent is to reduce mission planning time, develop more efficient mission plans and provide commanders with information needed to make resource decisions based on trade-offs. Specific points of contact concerning the sponsorship of this research include Mr. David Zann and Mr. Derryl Williams (AFRL/IFSC).

This research is also supported by a branch of the AFRL sensors applications and demonstrations division (AFRL/SNZ), specifically, the Virtual Combat Laboratory (AFRL/SNZW). The Virtual Combat Laboratory conducts advanced development field and flight test demonstrations and evaluations. They maintain a UAV simulation model along with a suite of visualization tools. This research continues the ongoing relationship between AFIT/ENG and AFRL/SNZW in which both parties share information on and enhance the capabilities of UAV swarm simulations. The specific point of contact is Mr. Mike Foster (AFRL/SNZW).

1.5. Thesis Overview

The remaining chapters assume the reader has a basic knowledge in the areas of combinatorics, evolutionary computation, self-organization, and unmanned aerial vehicles. Extensive references provided throughout provide the reader with more extensive background information in these areas.

Chapter 2 of this thesis defines and develops the concepts behind this research and gives a historical perspective on related research. Chapter 3 details the methodology used in this research and illustrates the high-level design strategies used in the development of a multi-objective routing algorithm and its use as inputs to VRP problem solvers. Chapter 4 shows the development of the specific implementation model used to achieve the research objectives. Chapter 5 describes the design of experiments used to evaluate the effectiveness and efficiency of the tools developed in this research. This includes the testing methodology, development of benchmarks and the scope of the experiments performed. Chapter 6 discusses the test results and evaluates the effectiveness of the research. Chapter 7 presents conclusions resulting from this research and makes specific recommendations for future swarm-based research.

2. Background and Historical Perspective

Mission Planning for swarms of autonomous aerial vehicles requires an efficient assignment of vehicles or sub-swarms to targets, a set of efficient, feasible paths for vehicles to follow, a set of swarm behaviors that allow the swarm members to reach their targets while maintaining their collective swarm properties, and a detailed simulation of the mission to ensure objectives are met. This chapter considers historical approaches to solving these individual problems as well as a discussion of ways to unify these problem domains into a comprehensive problem statement. A discussion on path planning is given followed by the development of the vehicle routing problem. The section concludes with various approaches to swarm modeling and their applications.

2.1 Path Planning

Path planning for air vehicles is a subset of a broader set of general path planning problems. All path planning problems and the algorithms used to solve them consist of some initial condition, objective, and a set of actions that completely connect the initial condition to the objective. However, there are many ways to specify a path planning problem. The method selected is often linked to the algorithm used to solve the problem. Two broad categories of path planning problems and approaches dominate the research. The first category defines the problem in what is known as a configuration space. Problem formulations of this type involve determine the set of desired actions (torques, rotations, and other forces) needed to move a system from an initial state to a goal state. The second category of problem formulations, trajectory spaces, involves generating a set of feasible trajectories to move a vehicle from an initial location to a goal location through a defined space. In this section a historical view of path planning problems of both configuration and trajectory spaces is presented.

2.1.1 Configuration Spaces. Historically, configuration space-based (C-space) problems have been applied to robotic motion. However, research has also used this domain to define problems for general 3-D motion. This section develops the notion of a C-space and examines some of the problems and algorithms that have been created in this domain. A Configuration Space, $\mathcal{C}$ [30] is defined as the position (or configuration) of an object completely determined by a single point having *n* independent parameters as coordinates. The space $\mathcal{C}$, is divided into two general subspaces, $\mathcal{C}_{\text{obs}}$ and $\mathcal{C}_{\text{free}}$. Configurations which are invalid because of constraints violations or collisions belong to $\mathcal{C}_{\text{obs}}$. Those configurations satisfying constraints and resulting in collision-free states belong to $\mathcal{C}_{\text{free}}$. In the configuration space, the path planning problem consists of finding a continuous curve (representing a path for a single geometrical point) that connects the points representing the initial and the final configuration of the object, and lies entirely within $\mathcal{C}_{\text{free}}$. In figure 2.1, [34] a 2-jointed robot arm is placed among a set of obstacles. The arm's joints can be rotated through a set of angles (q0, q1). An assignment of these angles results in either a valid configuration if the rotation of the joints does not result in a collision, or an invalid configuration, $\mathcal{C}_{\text{obs}}$, if the rotation causes the arm to collide with an obstacle.

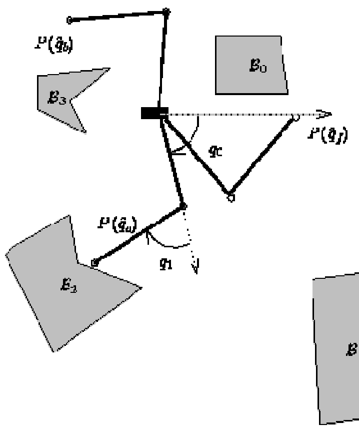


Figure 2.1. A Two DOF Arm in an Operational Space with Obstacles [34].

The configuration space for this problem is given in figure 2.2. The axes represent possible angular rotations of the joints q_0 and q_1 and the space contains valid and invalid sub-spaces corresponding to the assignment of these angles. A valid region of the configuration space represents the set of solutions to the problem.

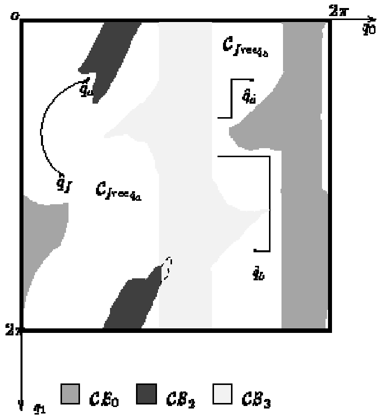


Figure 2.2. Configuration Space Corresponding to Figure 2.1 [34].

Configuration space-based problems have been solved using two main approaches: retraction methods, and decomposition methods [34]. In a retraction method, the dimensionality of the problem is reduced by considering sub-manifolds in the configuration space. A manifold is a term used to describe a region such that for every point, there is a set of neighborhood points that are topologically similar [66]. In a decomposition method, a characterization of the search space is partitioned into regions that are free of obstacles from which the search is conducted. These methods are complete but it has been shown that the retraction or decomposition of a search graph is an NP-complete problem [6]. Thus, these methods are inefficient for high-order dimension problems.

Gradient-based approaches have been used successfully for path planning in holonomic systems [7] [35] [65]. A holonomic system is one where if the system is returned to its original configuration then it also returns to its original position. An example of this would be a car on a track. Gradient-based methods include artificial potential fields. A naive artificial potential field is created by determining the distance to goal from each grid point in the search space. To traverse the field, the vehicle moves down its gradient to the lowest-values adjacent grid point. More detailed potential fields have been created as in the work of Batavia [4]. In his algorithm Batavia builds a potential field that combines the distance metric with a traversability metric that encodes the degree of difficulty a robot would have moving through each grid point. This level of difficulty ranges from impassable, as in the case of a solid obstacle in the cell to smooth. This difficulty information is combined with the distance field to make a single artificial potential field that represents the assumed cost of reaching the goal more completely. One difficulty with this approach is its scalability. Because every grid in the search space must be read to build the field, if the linear dimension is doubled, than the search space is quadrupled. Extending the method to three dimensions would cause an 8-fold increase in the search space.

Randomized tree-based methods have been successful in solving planning problems for non-holonomic systems [26] [29] [41]. Pure tree search methods are generally not viable for solving NP-complete problems because they are exhaustive. Randomized tree-based methods are probabilistic or probabilistically complete depending on how they are used. LaValle [29] developed a concept called Rapidly-Exploring Random Trees. Rapidly-Exploring Random Trees (RRTs) is a randomized data structure designed for a broad class of path planning problems. RRTs have been used successfully in robotic motion planning, wheelchair feasibility routes [29], Mars Rover motion software, and numerous pharmaceutical applications. According to Kuffner [25], RRTs alone are not sufficient to solve path planning problems but can be incorporated into other path planning algorithms as an efficient local search tool.

A generalized RRT begins with a single node, q_{init}. The algorithm runs through a specified number of iterations in which a new point q_{rand} is chosen from a random distribution, the nearest neighbor q_{near} in the tree to that point is identified, a point lying along the line between q_{rand} and q_{near} at a distance r from q_{near} is chosen as a candidate node in the graph. If this point lies in an obstacle-free area, then this node is added to the graph forming a new configuration. A novel aspect of the RRT algorithm is its method of searching with a bias toward unexplored regions of the search space. The key to understanding how this is done is by examination of the nearest neighbor operator and the Voronoi diagram associated with the graph. A Voronoi diagram partitions a set of vertices into regions such that any space within the region lies closer to the vertex contained within the region than it does to any other vertex in the graph. Imagine a small set of points clustered at the center of a search space. Within the center of the cluster, the Voronoi regions would be small. On the frontier of the cluster, Voronoi regions would be large and extend out to the limits of the search space. With the nearest neighbor operator, it is more likely that a random point selected is nearer to one of the larger Voronoi regions along the frontier than it would be to a smaller interior region.

Several successful path planning methods that use RRTs have been developed. RRT-Connect is one variant of the RRT in which two trees are developed, one growing out from the initial node and one growing out from the goal node. After the trees are allowed to grow, a connect algorithm is run to try to connect the two trees. At this point, a solution to a single-query path planning problem is satisfied [25]. The authors of RRT connect acknowledge that the algorithm performs poorly in areas with many obstacles.

Sampling-based methods of path planning have employed RRTs as an element of the overall algorithm [26] [41]. In [41], a probabilistic roadmap planner (PRP) is combined with an RRT local planner to create what is called the Sampling-based Roadmap of Trees (SRT). The PRP is a global planning tool that samples random configurations from the set of all collision-free paths. RRTs are then used to explore these initial configurations by building trees from nodes found in the PRP. The idea of a roadmap is that sections of route in a single query can be reused for later queries. The authors maintain that RRTs are chosen as one possible tool for the local tree search and that similar success has been achieved with Expansive Space Trees (ESTs) [29].

Another RRT-inspired planning algorithm is Path-Directed Subdivision Tree Exploration Planner or PDST-Explore [26]. The algorithm deliberately chooses a non-uniform, directed tree search to avoid searching far-off areas of the search space. RRTs, with their bias toward unexplored regions of the search space, search the four corners of a grid equally without regard to the location of the destination with respect to the direction of the current location.

Simulated Annealing (SA) has also been applied to solve a class of path planning problems known as mobile manipulator path planning problems by searching the configuration space [8]. The SA approach searches ordered sets of vectors applied to robots to perform their tasks and move through their workspace. This approach requires complete knowledge of the terrain and is suitable for well-known environments such as an automated production line.

A hybrid approach combining the PRM and SA was developed by Sanchez [50]. This method uses configurations generated by Kavraki's PRM and from these, used SA as a local search to find optimal paths. SA uses the concept of "local neighborhoods to restrict the search space at any particular time. The neighborhoods in the context of configuration spaces were defined in terms of the maximum allowable change of a given parameter in a given configuration to reach another configuration.

2.1.2 Trajectory Spaces. Unlike configuration spaces which consider transforming states to reach a solution, searches in the trajectory space involve creating a single path between the start and the goal node which may or may not be feasible. The path is refined and adjusted until a feasible and low cost path is found. Many path planning algorithms have solved problems with searches in the trajectory space.

In [3] a straight line segment joins the initial and final positions. The path naturally crosses forbidden regions. The algorithm progressively reduces the intersection of the path with obstacles. Each iteration creates a randomly-generated sub-manifold containing the current path. It is then discretized on a grid and explored using dynamic programming. The length of intersection of the path and the obstacles is used as the cost function in the dynamic program.

Other trajectory space searches have been done using evolutionary computation (EC). Inspired by the success of EC in solving many classes of problems, EC-based path planners have been developed. Many different problem representations have been considered. In [67] the problem is stated simply as: given a vehicle and a description of the environment, plan a path between two specified locations which is collision-free and satisfies certain optimization criteria. The optimization criteria given include: minimizing the path length, creating a smooth trajectory to avoid sharp turns, and maintaining a safe clearance between the paths and objects. The Evolutionary Planner/Navigator (EP/N) was designed by Xiao and Michalewicz to solve this problem using an evolutionary approach. In their work, each solution or chromosome is given as a linked list of coordinate pairs and an additional value indicating whether or not a given pair is in an obstacle-free region or not. Every chromosome contains the starting point, goal point, and a variable amount of intermediate points representing a path from the starting location to the goal. Chromosomes within the population are evaluated with three fitness functions each representing one of the three stated objectives. The fitness of an individual is determined with a weighted sum of the three evaluation functions. The weights are given as parameters and are used to control the relative emphasis of one criterion over another. A robust set of mutation operators are used to improve the set of candidate solutions in the population. After a number of evolutionary cycles, the best path found is returned. Through an extensive set of experiments, Xiao has shown that the EP/N usually returns a solution that is not far from known optimum at a fraction of the time it would take to perform an exhaustive search. EP/N differs from traditional genetic algorithms in two important ways. First, it uses a non-uniform chromosome length. A disadvantage of this approach is that it constrains the selection of appropriate data structures. Further, many GA software packages are available to speed development through the reuse of common GA operators and data structures. Most of these such as the Genetic Algorithm Library (GALib) and the Genetic Algorithm Utility Library (GAUL) are not compatible with variable length chromosomes. AFIT's General Multi-Objective Parallel Genetic Algorithm (GENMOP) allows for variable-length chromosomes but is currently available only in two application-specific versions [24]. An advantage of having non-uniform chromosomes as it pertains to path planning is that it allows solutions to be completely specified without having to explicitly list every grid point or cell along the route. The second departure from traditional GAs in the EP/N is the use of heuristics in the genetic operators. In a typical GA, mutation operators randomly flip bits of the solution or interchange values between one portion of the solution and another. In a path planning problem where there is a necessary ordering of points along the path, such random operators are more often destructive rather than constructive. For example, consider the following sequence of coordinate points that represent a path from a start point (0,0) to a goal point of (75,75). P_i = {(0,0), (12,4), (32,23), (48,37), (52,50), (68, 70), (75,75)}. Now consider a common mutation operator, the swap. A swap of the points (12,4) and (68,70) would tremendously increase the length of the path. Heuristics can be used to carefully steer an algorithm away from such problems by injecting problem domain information into the algorithm. Xiao's use of heuristics constrains the addition, deletion, and alteration of points to avoid destructive changes that can occur. The parallel path planner presented in this research (Chapter 3) uses a similar problem definition, chromosomal structure, set of operators and evolutionary approach as is found in Xiao.

Sugihara presents a GA for path planning in [59] that adds weighting factors into the evaluation function. Unlike Xiao that treats obstacles as hard constraints, Sugihara distinguishes obstacles into two categories: pure obstacles and hazards. Using this approach to defining the problem, obstacles are avoided at all costs but hazards are allowed albeit at an increase to the weighted cost of the path. The obvious advantage to this problem formulation is that imperfect solutions (those that intersect hazards) can provide significantly shorter paths than perfect solutions (those that are hazard free). In creating the problem instance, a planner could carefully decide between areas of mandatory avoidance (such as a no-fly zone, international border, etc) and areas that should be avoided or have minimal path intersection (such as enemy radar zones). With this problem domain knowledge, the particular problem instance can be tuned so that the algorithm returns a desirable path. Sugihara uses a more traditional GA than Xiao. Chromosomes are fixed length binary strings which encode a path that is *x*-monotone or *y*-monotone. The path consists of points such that each point is directly adjacent to the points after and before it. Mutation operators then alter the distance and direction of these points randomly without constraint. A limitation of the form of the solution is that the adjacent cell requirement results in a series of turns that are always 0, 45, or 90 degrees. While ground vehicles may be able to handle this zigzag motion, such a path would not be appropriate for air vehicles which have defined limitations on their turn rates.

Castillo reformulates Sugihara's work into a multi-objective problem and solves it using a multi-objective evolutionary algorithm (MOEA) [9]. In the single objective problem, a weighted sum of costs is used in a single objective function. In the multi-objective version, two objective functions are defined: cost and risk. The cost function measures the length of a candidate path while the risk function measures the amount of intersection a path has with a hazardous region. The algorithm returns not a single solution but a set of Pareto Optimal solutions with which a decision maker can chose an optimal path for a desired level of risk. In addition to using two separate cost functions, MOEAs have other differences from their single-objective counterparts. First, MOEA's tend to be a more effective with an elitist selection strategy rather than roulette wheel or tournament selection strategies which are commonly found in single-objective EAs [Cello and Lamont]. A comparative study among various MOEAs found that elitism converges more quickly and preserves good solutions [69]. Second, the use of adaptive mutation strategies, such as hyper-mutation, is more difficult to apply to multi-objective functions [9]. Hyper-mutation is a technique that varies the mutation rate as the solution quality of the fitness function decreases in successive generations [10] [15]. Since the objectives in an MOEA often compete with one another, it is difficult to respond to a drop off in solution quality. The third difference follows directly from the first two and that is the selection criteria cannot be biased toward improvements in either of the objective functions individually. With respect to path planning for air vehicles, Castillo's algorithm shares the same problem with Sugihara's, namely paths must be represented in terms of adjacent cells which leads to sharp, un-navigable routes.

Path planning is more constrained for UAVs than for mobile robots. While smooth trajectories are a priority in [67], routes produced by robot path planning algorithms would not in general be suitable for air vehicles due to the dynamic constraints imposed by the flight characteristics of the aircraft. Nikolos [39] offers an alternative path representation: B-spline curves. Developed by Schoenberg in 1946 and computed via recursion in 1972 by Carl de Boor [17], B-splines are piecewise polynomial building blocks that are convolved recursively to form complex curves. They can be completely specified using only a few control points called "knots." This compactness makes them suitable for use in evolutionary algorithms.

In his research Nilolos evolves candidate paths for UAVs by randomly joining sets B-Spline curve segments. The chromosomal representation of a solution consists of the control points of the individual segments which are evolved by standard recombination and mutation operators.

Mittal and Deb [37] extended Nilolos' work by creating a multi-objective version that optimizes both path length and risk. In their formulation risk is given in terms of how close a given path comes to a known obstacle. They further constrain the search with the following requirements: (i) the path must be collision-free with respect to solid obstacles; (ii) the angle between two successive segments must be below a specified threshold; and (iii) The maximum height of any point in the path must not exceed a specified upper limit. The last constraint is given so that the UAV remains closer to the ground so as to avoid detection by an enemy.

While B-splines can completely specify a path, they offer challenges to manned flight such as how would these curves be given to a pilot such that he or she could actually fly them? Typically, a pilot is given a set of waypoints to fly which when mapped out, are shown as a set of continuous line segments. Naturally, the aircraft flies a curved path when turning, but this curvature in the flight envelope is taken into account by specifying the turns in the route with sufficient margin for error. The distinction between a waypoint that must be reached and one that the pilot must come within a certain distance of is made in the flight plan and is called the "flyover" variable in the route specification. The paths given as B-splines would have to be approximated into line segments and then re-evaluated to ensure these new routes meet the turning constraints of the aircraft. If the difference between the original path and the discretized segment set is significant, then any optimality assumed in the original path cannot be guaranteed in the modified path. If this algorithm is to be applied for UAVs only, then the UAV mission controllers and designers are left with the task of computing all of the necessary control parameters so that the vehicle can fly the exact curve.

In this research, paths are specified in line segments with restrictions on the degree of turn to ensure the path is navigable. Further, the concept of terrain masking which was loosely developed in [37] is extended with a complete terrain masking algorithm developed by Air Force Research Laboratory. The algorithm determines the maximum altitude (AGL) of an aircraft at a particular point such that at or below this altitude it is out of sight of a known threat. This is known as inter-visibility. In addition to remaining out of sight of known threats, the terrain masking algorithm seeks to minimize the vehicle's exposure to unknown threats. This principle is known as "hidability." It calculates the number of nearby points from which a vehicle is visible at a given altitude over a given point. Figures 2.4 and 2.5 illustrate the principles of inter-visibility and hidability.

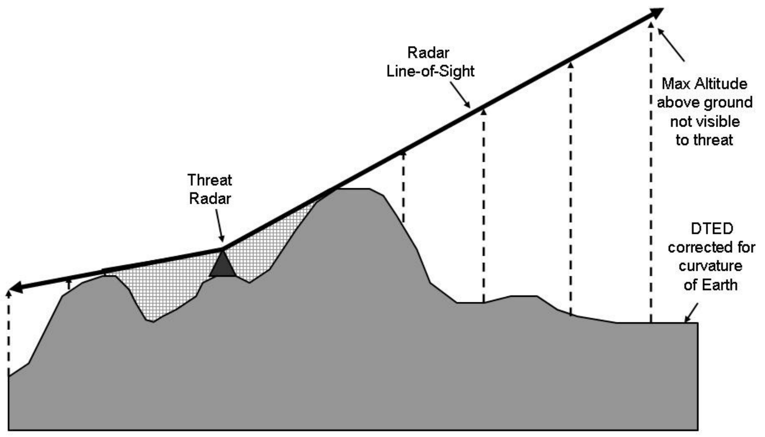


Figure 2.4 An Illustration of Intervisibility.

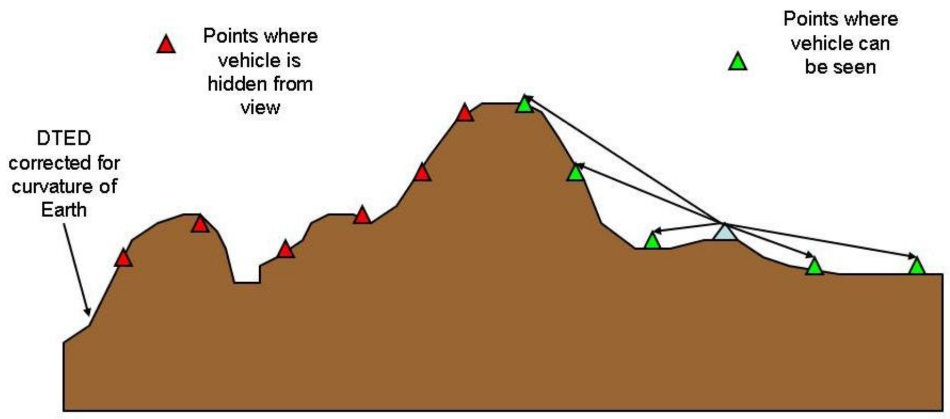


Figure 2.5 An Illustration of Hidability.

2.2 The Vehicle Routing Problem

The vehicle routing problem (VRP) can be defined generally as the task of optimally assigning a set of vehicles to visit a set of targets such that the cost of the assignment is minimal. It is an extension of the traveling salesman problem which is one of the most studied hard combinatoric problems of all time [2]. This section defines the VRP as a variant of the TSP, looks at evolutionary approaches used to solve it, and finally, connects the path planning problem to the VRP via a simple reformulation of the path planning problem definition.

2.2.1 The TSP and VRP defined. The goal of the TSP is for a person to visit every city in his or her territory exactly once and return home while traveling the shortest distance possible [38]. The problem is defined as follows: Given a complete undirected graph $G = (V, E)$ that has a non-negative cost $c(e)$ associated with each edge $e \in E$, find a tour (an ordered set of all the cities in the graph G) such that for all nodes $u, v, w \in V$, $c(u, w) \leq c(u, v) + c(v, w)$. This problem has been proven to be NP-complete [12]. The TSP is a historically significant problem because it is a generalization of many real-world problems in transportation, shipping, and routing [38]. One real-world application of the TSP is the vehicle routing problem which is also a generalization of an entire class of routing problems. The capacitated vehicle routing problem (CVRP) is the simplest version of the VRP. In the CVRP, a set of customers require a discrete amount of service. There are a set of vehicles each with a known capacity. The task is to optimize the assignment of vehicles to the customers so that the total distance traveled by the vehicles is minimized. In most versions of this basic problem, there is the added constraint that no vehicle can make a partial delivery of less than one unit of demand to a customer to be finished later by a second vehicle [62]. It is from this version of the problem that Russell defined his UAV routing problem [49]. Because the CVRP is an extension of the TSP it is also NP-complete. The UAV routing problem consists of a set of targets *L*, a set of UAVs *V*, the set of traveling costs *Q*, the set of routes *G*, a distance function δ, a capacity function γ, and a demand function α. The formal definition specified by Russell is as follows:

given $L : \forall l_i \alpha(l_i) \geq 0, i > 0;\quad \alpha(l_0) = 0$ and $V : \forall v_i \quad \gamma(v_i) = k, k > 0$

compute: $Q : q_{ij} = \delta(l_i, l_j)$ and $G : g_k = \{l_0 \bigcup L \times L \bigcup l_0\}$

subject to: $\sum_{l \in G} \alpha(l) = \sum_{l \in L} \alpha(l)$ and $\bigcup l \in g = L$ and $\bigcap g \in G = l_0$

minimize: $\sum_{k=1}^{|Q|} Q_k$

The particular application Russell addressed was a swarm of heterogeneous UAVs required to deliver an amount of munitions or reconnaissance payload to a static set of targets. In his work Russell does not model threats as areas along the route, but rather by associating a threat level associated with each target. In this way an additive cost function that includes risk as well as distance is generated for every link in the graph representing the battle space. A shortcoming of this approach is that no attempt to avoid risk is made. Further, the direction of approach has no impact on the amount of risk associated with the target. Clearly this is an oversimplification. If risk is to be considered as an optimization criterion, then the routing of a swarm must include a path planning algorithm that minimizes risk as well as distance. The next section describes Russell's approach to solving the UAV routing problem.

2.2.2 Genetic Vehicle Representation. Originally developed in [61], Russell uses Genetic Vehicle Representation (GVR) in his genetic algorithm for UAV routing. GVR consists of a novel data representation, and original *mutation* and *crossover* operators. The key element of the encoding is that solutions explicitly provide the number of routes and their locations without significant decoding. For *crossover*, two individuals C_1 and C_2 are chosen for selection. A sub route r from C_1 is inserted into a copy of C_2. The position of the insertion is such that the distance between the insertion location and the first location r is minimized. This is a departure from traditional crossover operators because it brings problem domain information into the operator. It is clear that inserting points at random locations without regard to the structure of the tour would be destructive more often that constructive.

There are a total of four *mutation* operators used in GVR. In the first, *swap mutation*, two locations within a solution are exchanged. As the solution contains routes for different vehicles, the swap may change locations within a route or between two routes. In either case, this operation maintains the same number of locations between the original solution and the resulting one. The second mutation operator is *inversion*. This operator takes a sub route of random size from a solution and reverses the order of locations. This operator was found to be destructive in nature and is used with low probability in Russell's implementation. The third mutation operator is *displacement*. Similar to crossover, a sub route is removed from a solution and moved to a new location. Unlike the crossover operator however, no effort is made to place the sub route in an insertion point that minimizes distance. Russell describes this operator as "relatively destructive" and uses it with low probability. The final mutation operator in GVR is *insertion*. This operator is a special case of displacement in which the size of the sub route is one. This property makes the operator less destructive since only one route is affected. To validate his work, Russell tested his algorithms against the Lin-Kernnigan Traveling Salesman Problem Library benchmarks (TSPLIB) [2].

For handling feasibility concerns, Russell developed a set of experiments to compare among repair functions and static and dynamic penalty functions. Empirically he showed that for his problem domain, that the use of a repair operator led to faster convergence and thus shorter runtimes than using either static or dynamic penalty functions. He cites as a possible explanation for this, the difficulty in integrating sensitive problem-specific information into the creation of a penalty function. A repair function on the other hand is at least straightforward if not generally superior.

2.2.3 Linking the Path Planning Problem to the UAV Routing Problem. Despite the many definitions of the path planning problem in the literature, nearly all of them structure the problem as the minimization of cost in traveling between a start and goal node. As a result, the output from any good path planning algorithm can provide the set of link weights for any algorithm that solves the routing problem. In Russell's GVR implementation, the link costs are calculated using a combination of the path length and the risk associated with the target node. In a classic TSP problem where the nodes represent distant cities, this is an adequate approach. Any cost associated with a vehicle having to turn around within a city is abstracted away due to the low relative cost of such a maneuver compared to the distance between the nodes. In a UAV mission however, this may not be the case. If the locations or targets in a UAV routing problem are confined to a single city, the distances between the targets may be as close as a few aircraft turn radii. In this case, turns which can be made abruptly in a large scale problem now become factors in the minimization of costs for a tour. A point-to-point link cost is no longer sufficient to represent the true cost of the link. In order to account for the cost associated with turning, the cost of a link must consider the heading of the aircraft when approaching and leaving the various target nodes. By reformulating the path planning problem in terms of target triplets, with the goal of minimizing the cost of traveling from a starting point p_0 to a destination point p_f through an intermediate point p_m, a full account of the added distance incurred by making feasible turns can be made. A path planner with this capability then takes as its input three points representing targets, terrain and threat data corresponding to the planning area and return a path containing a complete set of points representing a path that travels through the three points while negotiating threats and minimizing cost.

2.3 AFIT UAV Swarm Simulator

The AFIT UAV Swarm Simulator is a Parallel Discrete Event Simulation (PDES). Based originally on Reynolds' Distributed Behavior Model for flocking, the simulator was developed by Kadrovach [23]. Corner [14] ported the model from a single-processor Windows platform to a parallel Linux-based Beowulf cluster. The simulator appeared twice at the Winter Simulation Conference [13] and [48]. This section looks at the historical development of the model and a description of the architecture under which it was developed.

- *Reynolds' Distributed Behavior Model.* In his model, Reynolds defines three behaviors or rule sets to which individuals must adhere to maintain a flock or swarm [45]. The first of these is collision avoidance or separation. This behavior has the highest priority among the three. It involves steering to avoid crowding or collision between flock mates. Velocity Matching or Alignment is the second most important behavior. Adherence to this rule requires each flock member aligning itself toward the average velocity of the flock. Finally, flock centering or cohesion is lowest priority behavior and involved steering toward the average position of local flock mates. Figure 2.6 [45] illustrates these principles.

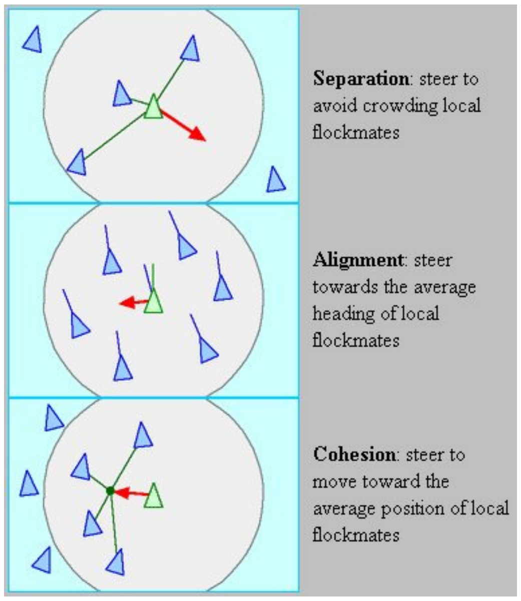

Separation: steer to avoid crowding local flockmates
Alignment: steer towards the average heading of local flockmates
Cohesion: steer to move toward the average position of local flockmates

Figure 2.6 Reynolds' Distributed Behavior Model [45].

Together, these three behaviors provide a basic rule set for modeling a swarm or flock. Since Reynolds' publication in 1987, many swarming models have been developed which incorporate these principles. Among these models are Particle Swarm Simulation, Physical Robots, Synchronized Multi-Point Attack, Self Organized Behavior Swarms, the Kadrovach Model, and the Milam Model.

- *Particle Swarm Simulation (PSS).* Based on Particle Simulation Codes (PSC), particle swarm simulation models use plasma ion interactions as their inspiration. PSCs have standard force equations and equations of motion. Additional model features of the PSS include center-of-mass, swarming force, friction, dissipation, aerodynamics, and gravitation [63]. These various forces and rules of the PSS are summed into single vectors of motion.

- *Physical Robots*. Described in [32], an agent-based model is developed with the following behaviors: avoidance, following, aggregation, dispersion, homing, and wandering. Agents also have goal knowledge which contributes to the development of successful behavior.

- *Synchronized Multi-point Attack.* Lua developed a model in which a swarm converges to a single target and attacks [31]. In this model, an orbit behavior is defined under which a vehicle begins to circle around a target and wait for other swarm members to join in. Once the orbit density reaches a critical level, the model moves the swarm into attack mode allowing the members to attack in mass from different directions.

- *Self Organization of UAV Swarms.* Price developed an extensive swarm behavior model in which behavior archetypes are selected using a perceptron and evolved using genetic programming [42]. The swarms' members follow numerous rules which allow them to seek out unknown targets in an unknown region, gather to a critical mass, and engage the target.

- *Kadrovach Model.* The AFIT swarm simulator is based on the Kadrovach model which was developed upon the more general Reynolds model. The model focuses on communication and formation stability. Communication networks require scalability. To this end, Kadrovach limited the communication among nodes to nearest neighbors. A UAV's nearer neighbors shadow or block other neighbors within 30 degrees. With this restriction, swarms tended to form hexagonal configurations. In addition to formation stability, this rule also achieved the desired scalability. As swarm size increases, the number of communication pathways required per node remains constant. In both Kadrovach's initial single-platform implementation and Corner's subsequent parallel implementation, the limitation on node-to-node visibility was only simulated. In Kadrovach's model UAVs had global knowledge of every other UAV's position. In Corner's parallel PDES implementation, UAVs subscribed to all other UAVs. While their motion vectors were only affected by nearest neighbors, scalability was poor as network message traffic increased significantly with increasing numbers of UAVs.

The physical behavior of the Kadrovach model was guided by combining Reynolds' three rules into just two: attraction and alignment. The attraction rule causes UAVs to adjust their motion vectors to steer toward one another whenever distances between UAVs exceed a given threshold. Conversely, the same rule forces nodes to steer away from one another when their distances decrease below a given threshold. The alignment rule is implemented just as described in Reynolds.

- *Milam Model.* This model focuses on control aspects of UAV swarms [36]. A genetic programming algorithm is used to evolve UAV velocities throughout the life cycle of the model. This model successfully moves a swarm of UAVs through a 3-dimensional space among a set of targets while preserving the integrity of the swarm.

- *Reynolds Behavioral Hierarchy*. In his later work [46], Reynolds defined a more complex behavior hierarchy for autonomous characters. Complex goal-oriented behaviors at the top of the hierarchy are produced by aggregations of lower level behaviors. Figure 2.7 illustrates this hierarchy.

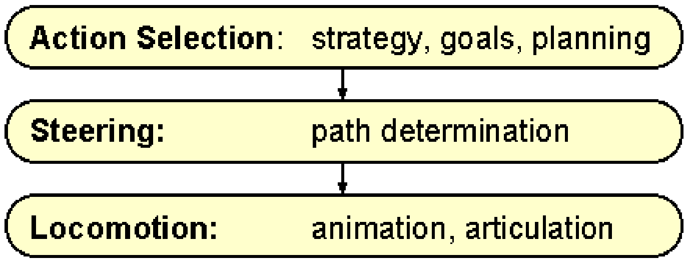

Action Selection: strategy, goals, planning
Steering: path determination
Locomotion: animation, articulation

Figure 2.7 Reynolds' Behavior Hierarchy [46].

These behaviors are best understood through example. Imagine a redundant mobile ad hoc sensor network, with the ability to self-repair. A sensor detects a nearby sensor going offline. This represents an undesirable change in the communications environment. The node repairs the hole in the network by "deciding" to reposition itself to fill in the gap. This is an example of an action selection behavior. To achieve this new goal of closing the hole, the mobile sensor must plan its path of motion to travel to the desired location while avoiding obstacles. This is a steering behavior. To steer along this path to the hole, a set of low-level locomotion behaviors such as thrust and turning are required. Thus complex behaviors are defined and achieved through the careful application of low-level behaviors.

Within this framework, Reynolds developed a complex model for 3-D autonomous animation. This model was implemented with video games in mind and is used in Sony's Playstation ™ game system. The physical model used to represent a vehicle was based on a point mass model. Reynolds acknowledged that this model overlooks turn radii and moment of inertia. The vehicle model consists of a mass, position, velocity, maximum force, maximum speed, and an orientation. The orientation is given as a set of N-basis vectors and is therefore suitable for both ground and air vehicles.

With the point mass vehicle model in place, the behaviors associated with locomotion hierarchy act directly on its vectors. The control signals which generate the locomotive behaviors are communicated through the steering behavior. This middle hierarchy was fully developed to improve on the three-behavior model of his earlier work. Behaviors developed under this hierarchy include: seek, flee, arrival, pursuit, offset pursuit, path following, obstacle avoidance, and containment. Seek is the pursuit of a static target. It acts to steer a character toward a particular position. Flee is the inverse of seek. It steers the character so that its velocity is radially aligned away from a fixed location. Figure 2.8 illustrates the behaviors of seek and flee.

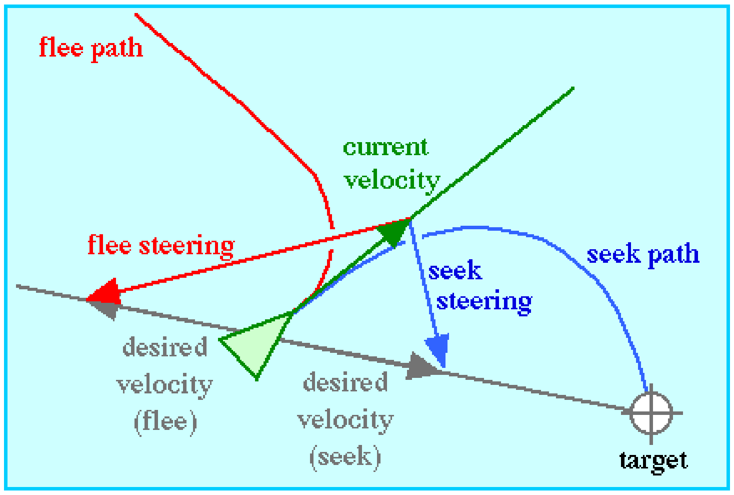


Figure 2.8 Reynolds' Seek and Flee Steering Behaviors [46].

Pursuit is like seek but the added factor that the target is moving. This behavior requires not only knowledge of the target's velocity vector, but also the capability to predict the targets future velocity. Evasion is the opposite of pursuit i.e. the character is steered away from the predicted future location of the moving target. Figure 2.9 illustrates the pursuit and evasion behaviors.

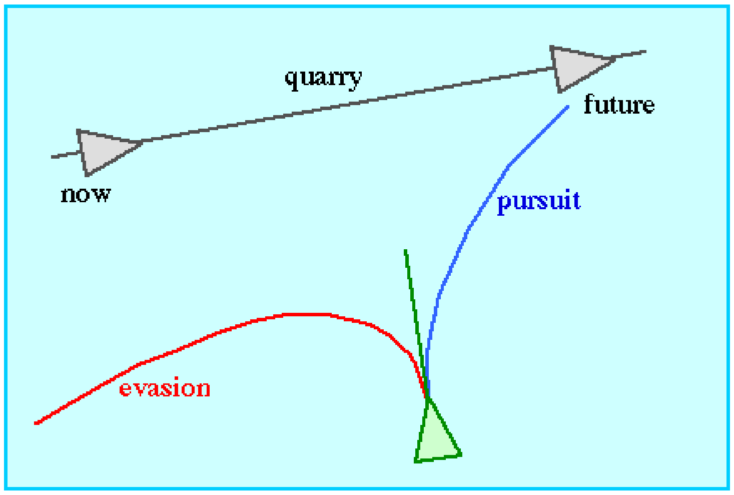


Figure 2.9 Reynolds' Pursuit and Evasion Steering Behaviors [46].

Offset pursuit steers a path to come within and maintain a fixed distance from a moving target. Arrival is the same as seek when there is a significant distance between the vehicle and the target. However, arrival slows the vehicle down as it approaches. This behavior ends with the vehicle at a zero forward velocity and a position coincident with the target. Figure 2.10 illustrates the arrival behavior.

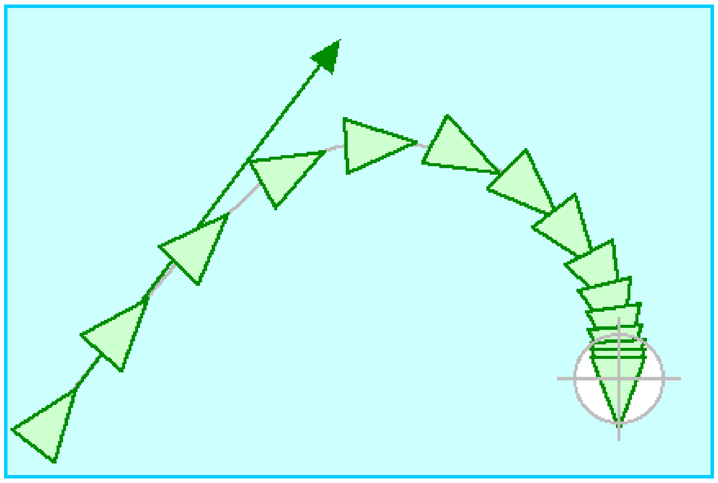


Figure 2.10 Reynolds' Arrival Steering Behavior [46].

Reynolds' extensive behavior hierarchy addresses many of the requirements a swarm must meet in order to be able to follow feasible paths to targets. The behavior set is rich and requires a complex set of individual members to execute. In this research, some of these capabilities are created offline. For example, feasible paths are generated by the path planning module and assigned to swarm members thus relieving them of burdensome computational requirements. At the strategic level of planning, the assignment of sub-swarms to target sets is also performed offline allowing decision makers, rather than swarm members themselves, to better guide the behaviors of the swarm to meet the objectives.

2.4 Summary. This chapter discusses various formulations of the path planning problem and some algorithms used to solve them. The vehicle routing problem is then defined with a look at Russell's genetic algorithm used to solve it. Finally, swarm behavior models are explored with emphasis on the existing AFIT Swarm model. These three research areas motivate the design of the enhanced AFIT UAV Swarm Mission Planning and Simulation System. Chapter 3 provides the high level design of this system as well as designs of its three components: the parallel path planner, the modifications required of the vehicle router, and the enhanced swarm behavior model.

3. High Level Design

This chapter develops the higher-level design aspects of the AFIT UAV Mission Planning and Parallel Simulation System and the associated design objectives of this research. The system consists of three principal components: a parallel path planner, a vehicle router, and a simulation and visualization engine. The problem and algorithm domains of each of the problem's components are presented as well as the strategy developed to integrate the algorithm domains into a comprehensive system.

3.1 Design Objectives

The development of a comprehensive UAV mission planning system consists minimally of an efficient assignment of resources to targets, an effective means to create vehicle trajectories that minimizes risk to the resources and mission cost, and a behavior model that produces swarm behavior without degrading the other capabilities. This section compartmentalizes the above requirements into three design modules each with a defined problem domain and suitable algorithm domain. The routing module is enhanced from previous research; a formal description of its problem domain/algorithm domain mapping is given in [49].

3.1.1 Parallel Path Planner. The Parallel Path Planner is developed with two key objectives: create an efficient and effective path planner using a genetic algorithm, and create a flexible parallelization of the algorithm to allow for rapid generation of multiple paths for use in solving higher level optimization problems such as the TSP and CVRP [62].

- *Problem Domain of the Path Planner.* The specific path planning problem for air vehicles consists of the following: given a discrete operational space of size *n x m* units superimposed over a terrain grid $G \in (n-1) \times (m-1)$, a Location set L where $l_i \in n \times m \forall l \in L$ find a low cost path P^* from all $l_i \in L$ to all other $l_j \in L$ subject to the following: $\forall p_o ... p_n \in P, \Delta\theta(p_i, p_{i+1}) \leq 45^{\circ}$ where θ is the inbound heading at p_i. The restriction $\Delta\theta \leq 45^o$, ensures that the path remains flyable by an aircraft. Based on the grid spacing of 750 meters, the vehicle can safely navigate a 45-degree turn. The threshold is based on the restriction used by the Terrain Following Optimizer (TFO) of the Air Force Research Laboratory and is representative of the turn rate of the AC-130 gunship [58]. This turn restriction can easily be modified to suit other vehicle types that may have faster or slower turn rates. The term "cost" is a composite of individual objectives or measures of merit of a mission. In this research, five such measures of merit are defined. These objectives are the same as those used in the TFO of the Air Force Research Laboratory.

1. Path – The sum of the Euclidian distances of the route segments to include the turns that connect them.

2. Climb – The amount of climbing a vehicle must do in the course of flying a route in order to avoid terrain.

3. Terrain – The cost of exposure to unknown threats or the vulnerability associated with being "out in the open."

4. Detect – The cost associated with being exposed to enemy detection – a function of both distance and time.

5. Kill – The cost associated with being within the lethal range of an enemy air defense weapon – a function of range, time and the lethality of the weapon.

While the problem domain of the generalized path planner has no restriction on the size of the target set *L*, the target set is limited to three targets or locations per instance to maintain compatibility with the problem domain of the TSP/CVRP which is solved by the router.

- *Multi-Objective Problem Formulation.* When a problem has five different cost functions, it can be solved as an aggregate function that attempts to simultaneously minimize all parameters, or it can be solved as a multi-objective problem where the output consists of a set of non-dominated solutions along the Pareto front. An end user can select one of these solutions provided they are capable of deciding the appropriate level of trade-off between two competing objectives. An output consisting of a five-dimensional Pareto front however, would likely overwhelm the decision maker by providing him or her with more questions than answers. Fortunately, the measures of merit can be grouped logically into two categories: those that describe the cost of the path in terms of time and fuel consumption, (*path* and *climb*), and those that measure the risk of a given path (*terrain*, *detect*, and *kill*). Equations 3.1 and 3.2 define the grouping of the five problem objectives into two competing categories.

$$\Phi_{COST} = \alpha\Phi_{PATH} + \beta\Phi_{CLIMB} \tag{3.1}$$
$$\Phi_{RISK} = \delta\Phi_{DETECT} + \lambda\Phi_{KILL} + \omega\Phi_{TERRAIN} \tag{3.2}$$

Where {α,β, δ, λ, ω} are weighting factors associated with the relative importance of each parameter. In the TFO, these values are mission specific and are input by the user. The individual cost functions are:

Φ_{PATH} : The Euclidian distance between each point is summed over the length of the route.

$$\Phi_{PATH} = \sum_{i=0}^{f} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \qquad (3.3)$$

Φ_{CLIMB}: The sum of positive changes in elevation from each point to the next point.

$$\Phi_{CLIMB} = \sum_{i=0}^{f} \Delta z(p_i, p_{i+1})\delta \qquad (3.4)$$

where δ is 1 if $z_{pi+1} > z_{pi}$ and δ is 0 otherwise.

Φ_{DETECT}: The total linear distance through which an aircraft flies into the effective detection ring of an air defense radar. In this model, an inter-visibility database is calculated using the terrain and threat data sets. Inter-visibility accounts for areas in which terrain obscures the radar's line of sight to the aircraft. So while a typical threat ring over flat terrain might be represented visually as a circle, the inter-visibility database over a rough area would show the threat ring as a circle with shaded portions representing safe areas where the radar is ineffective. We define the detection penalty as a summation of the penalties associated with each point in the route such that in traveling from one point to the next, the length of the intersection between that segment and an effective threat area is a scalar multiple of the penalty. Figure 3.1 graphically depicts an instance of a route intersecting a detection area.

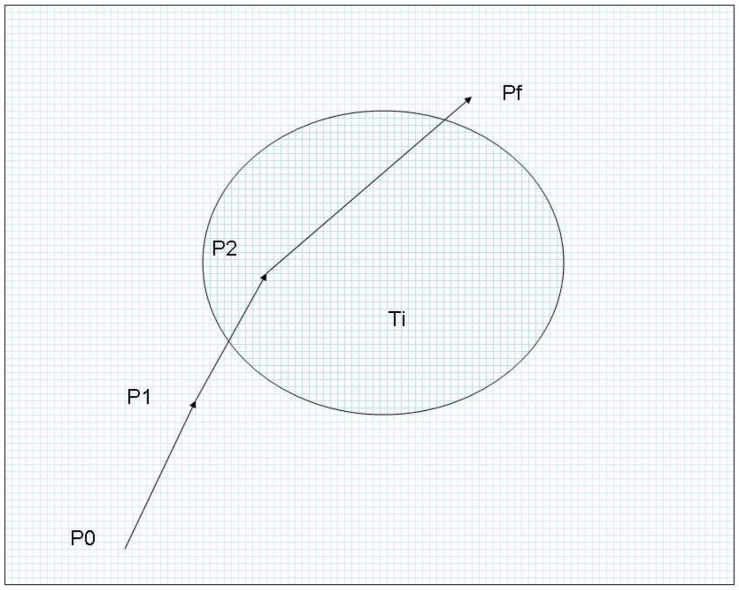


Figure 3.1 Example of Route Intersecting a Detection zone.

The formulation of the threat penalty is described as: given a battle grid (X,Y), a route {p_0, p_1, p_2, p_f}, where points between p_i and p_{i+1} are in (X,Y), and a threat detection zone $T_i \in (X,Y)$, we define the cost of traveling through a threat as the sum of the number of (X,Y) in *Ti* intersecting the route P.

$$\begin{aligned} &\sum_{i=0}^{\Delta x} \sum_{j=0}^{\Delta y} \delta(i,j) \text{ where} \\ &\delta = c_{INTERSECT}\, if\, (i,j) \in Ti \\ &\delta = 0 otherwise \end{aligned} \qquad (3.5)$$

$\Phi_{KILL:}$ The same formulation required for the detection cost function is applied to the kill cost function. The distinction between the two is that the effective kill radius of an air defense system is generally smaller than the detect radius. Information regarding the type and range of a given threat is either known or estimated a priori.

$\Phi_{TERRAIN}$: While many threats are known a priori, others are not. Therefore, the vehicle should remain out of sight as much as possible. The terrain metric measures the number of points in the grid from which a vehicle at a particular point can be seen. The overall terrain score is determined by summing the surrounding points from which the vehicle can be seen as it flies though each grid point along its path.

- *Algorithm Domain of the Path Planner.* Given the historical development of the path planning problem shown in Section 2.1.2, a multi-objective evolutionary approach was chosen as the framework to develop a path planner compatible with the AFIT CVRP router. The multi-objective approach allows for commanders to make informed decisions on mission trade-offs with respect to cost and risk. Although a multi-objective problem has many solutions, a router can only use a single solution for a given link to solve the CVRP. Therefore, in this design, only a single solution from the Pareto front of the path planner is passed to the router. The planner is still viable as a multi-objective decision making tool however. Smaller missions with limited target sets and a low number of vehicles can be optimized using individual instances of the planner for each vehicle. The output of the planner then provides a decision maker the cost/risk tradeoffs of each vehicle.

The multi-objective genetic algorithm of the path planner consists of the following elements: a population of candidate solutions, a defined chromosome structure of each candidate, a set of evolutionary operators which operate on the members of the population, a pair of evaluation functions to measure fitness of the solutions, an archived set of non-dominated solutions, and a defined period of evolution. Based on the work of

Xiao [67] a population size of 50 individuals was used along with an evolutionary period of 50 generations. Figure 3.5 below describes the high level flow of the evolutionary algorithm used in the path planner. The vector function $f_k(\vec{x})$ is the set of evaluation functions. In this algorithm, k=2 where $f_1(\vec{x})$ is the cumulative cost function and $f_2(\vec{x})$ is the cumulative risk function. The symbol g, is the period of evolution (50 generations).

Algorithm MOEA Planner algorithm

```
1: procedure MOEA_Planner($\mathcal{N}$, $g$, $f_k(\vec{x})$)
2:    Initialize Population $\mathbb{P}$ of size $\mathcal{N}$
3:    Evaluate, Rank (by dominance), sort Population
4:    Create archive population $\mathbb{P}_a$ from non-dominated members of $\mathbb{P}_i$
5:    for i in 1 to g do
6:        Select for recombination
7:        for j in 2 to $\mathcal{N}$ do
8:            Statistically select mutation operator $\Gamma_k$
9:             Mutate member j
10:       end for
11:       evaluate Population
12:       determine dominance rank within current population $\mathbb{P}_g$
13:       remove dominated members from $\mathbb{P}_a$
15:       add globally non-dominated members from $\mathbb{P}_g$ to $\mathbb{P}_a$
16:   end for
17: end procedure
```

Figure 3.2 High Level View of Path Planning Algorithm.

Note that the algorithm runs for a fixed number of generations. No heuristic was developed to terminate the evolutionary cycle once convergence of the solution was achieved. Further experimentation is needed to study the time saving benefits associated with early termination of the algorithm. The chromosome structure is similar to that used in [67]. A chromosome or candidate solution consists of an ordered set of points (x_i,y_i) which define a path from the starting point (x_0,y_0) to a destination point (x_f,y_f) through a midpoint (x_m,y_m). Additional information contained in each point includes elevation, the MSL altitude of the point; set clearance, the AGL altitude of the point; and heading, the direction of travel from the present point to the next point. Set clearance and altitude are used to calculate the amount of climb or descent needed to reach the next point as well as for terrain masking calculations. Heading is stored to ensure feasibility of the turns. The planner calculates the change of heading between points to ensure the turn rate is within the aircraft's limits. Figure 3.3 illustrates the chromosome structure of a candidate solution.

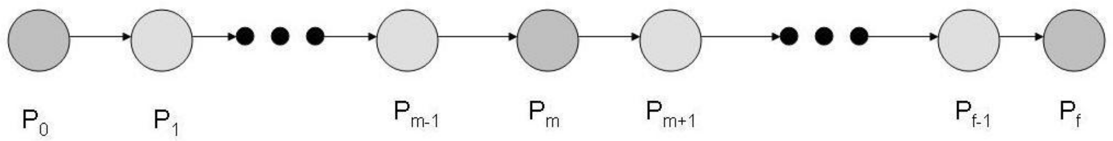


Figure 3.3 Chromosome Structure of the Path Planner.

During initialization, the population of candidates is created with each member containing the start, middle, and end points. An initial check is performed to ensure that the turn around the mid point is less than 45 degrees. If it is not, a modified convex hull algorithm (See Appendix B) is used to add additional points to the route such that no turn greater than 45 degrees remains. Once the route is repaired, a number of intermediate points are randomly added to the route. The number of points added is based on the distance between the three original points. During this process, the algorithm ensures that the change of heading between each point (excluding the starting point) is less than 45 degrees. Figure 3.4 defines the algorithm for population initialization.

Algorithm: Population Initialization of the Path Planner

```
procedure INITIALIZE($R_{init}$, $p_0$, $p_m$, $p_f$, $R$[POPSIZE] )
    $R_{init} = \{p_0, p_m, p_f\}$
    if | $p_m$ heading - $p_0$ heading| > 45 then
        REPAIR($R$[i])
    end if
    points to add before mid = distance ($p_0$, $p_m$) / 10
    points to add after mid = distance ($p_m$, $p_f$) / 10
    for i in 1 to POP SIZE do
        for i in 1 to points to add before mid do
            $R$[i].insert random point
        end for
        for i in 1 to points to add after mid do
            $R$[i].insert random point
        end for
    end for
end procedure
```

Figure 3.4 Population Initialization of the Path Planner.

Once the population has been initialized, it is evaluated using the cost functions described in section 3.3.2. In a single objective EA, a program need only maintain the current population. In a MOEA, the complete set of non-dominated points is maintained. A non-dominated point *P*, is one that has a fitness value for one objective function, F_1, such that no other solutions exist with a lower score (minimization problem) for objective function F_1 unless that solution has a higher score for another objective function. The set of non-dominated solutions is known as the Pareto front. To store the set of non-dominated solutions for this MOEA, a Pareto front archive is maintained.

To find the initial Pareto front points, each member of the population is compared to every other member based on the member's F_1 score, Φ_{cost} and by its F_2 score, Φ_{risk}. The population is first sorted by Φ_{cost}. A candidate $\boldsymbol{R}_i$ is added to the Pareto front if it meets the following criteria.

$$\forall p \in R, \neg \exists R_p \mid F_1(R_i) > F_1(R_p) \wedge F_2(R_i) \geq F_2(R_p) \qquad (3.6)$$

All non-dominated members of the population are added to the Pareto Front Archive. The population is then sorted by rank. The rank of an individual R_i, reflects the number of individuals in the population that dominate R_i. All non-dominated members of the population are assigned a rank of zero. All members dominated by a single solution are given a rank of one. Members dominated by two individuals are given a rank of two etc. Rank is the primary selection criterion used in the path planner. Dominance count is an alternative selection method. Dominance count is defined as the number of solutions in the population that a particular solution dominates. A drawback of using dominance is that points along the ends of the front tend to evolve out of the populations while crowding occurs near the middle of the front. Rank is therefore preferable to raw dominance count because greater diversity is maintained in the population.[24].

Once the population has been evaluated and ranked, selection is performed. Like other MOEAs, [9] [24] the planner uses an elitist selection operator. Elitist selection chooses the fittest members of the population. The use of elitism is common in MOEAs because the elitism preserves non-dominated individuals. The top half of the rank-sorted population is selected for recombination. Pairing of individuals is done randomly. Once paired, two offspring are created. These offspring occupy the places of the members not selected. Crossover is performed at the midpoint of the path. This ensures that the offspring remain feasible. During the crossover operation, the midpoints between two parents are exchanged. Since the underlying data structure (described in Chapter 4) is a linked list, the points beyond the midpoint are copied as well. The resulting offspring contain the points of one parent from the start of the path to the mid point, and the points of the second parent from the mid point to the end of the path. An illustration of the crossover operator is given in figure 3.5.

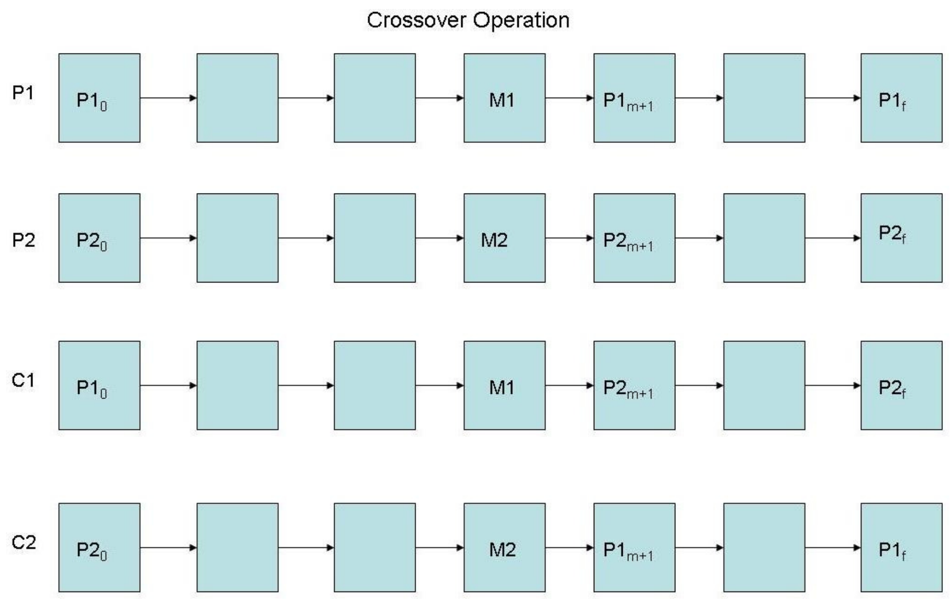


Figure 3.5 Crossover Operator.

Once the crossover operator has been applied, the population then undergoes mutation. The path planner uses three distinct mutation operators which are applied with equal probability.

The first mutation operator, M1 attempts to add a point between two existing points in the path. If the addition of the point results in an infeasible solution, then the repair operator is invoked to create additional navigation points. The sharper the turn created by the mutation, the more navigation points are needed to smooth the route. The repair algorithm generates a number of points proportionate to the change in heading caused by the infeasible point. For turns of just over 45 degrees, only two points are needed. For larger turns, as many as seven additional points need to be added.

Therefore, when the mutation operation adds a point between two relatively nearby points, resulting in an unfeasible route, the path cannot be repaired and the operation is cancelled. Figure 3.6 illustrates the Mutate Add operation on an arbitrary 4-point path segment.

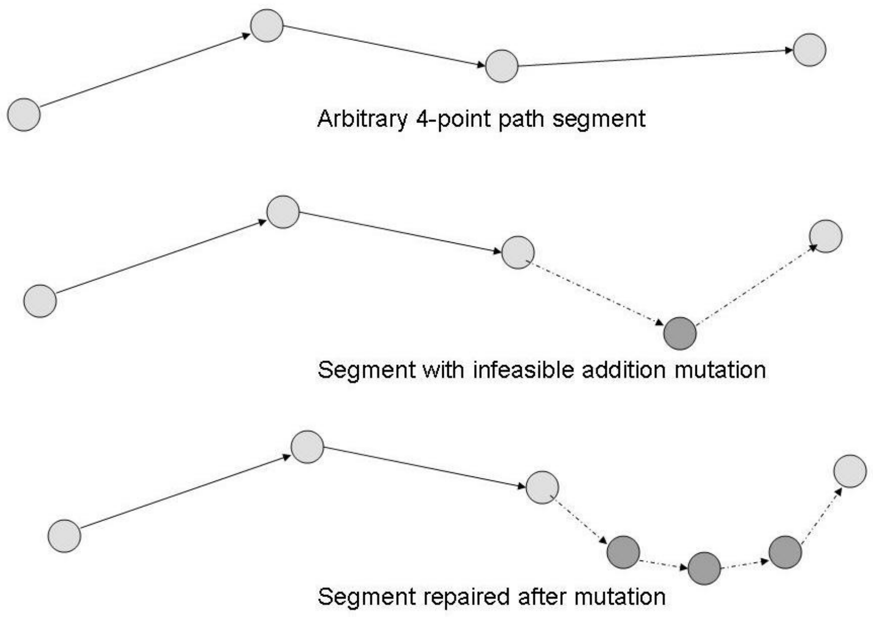

Arbitrary 4-point path segment
Segment with infeasible addition mutation
Segment repaired after mutation

Figure 3.6 Mutate Add Operation with Repair.

The second mutation operator, M2 attempts to delete a point between two points in the path. Again, if the deletion results in an infeasible path, the repair operator is called to add points which result in a smooth trajectory. Deletion operations naturally increase the distance between points. Therefore, the repair operator is usually able to add the points necessary to achieve feasibility. Nonetheless, feasibility of the repair operation is still validated and if the path cannot be repaired, the operation is undone. It is important that balance is achieved between delete and addition operations. When too few deletions occur, the resulting path has too many points and is more difficult to evolve.

When too few additions occur, the path tends to have very few points and the ability of the algorithm to minimize cost and risk is diminished. Because the deletion operation results in greater success, the addition operation is used with a slightly greater probability. An example of the delete mutation operation is given in Figure 3.7.

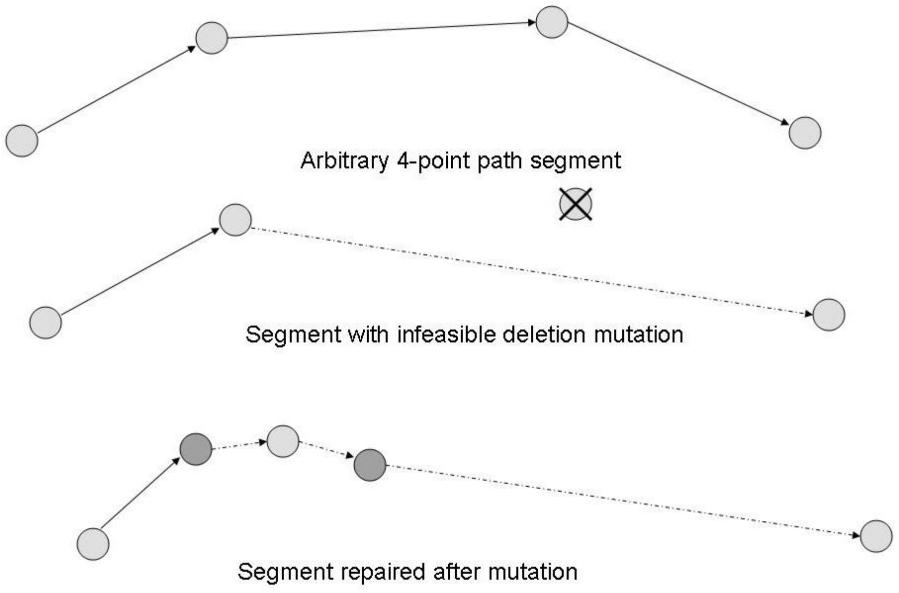

Arbitrary 4-point path segment
Segment with infeasible deletion mutation
Segment repaired after mutation

Figure 3.7 Mutate Delete Operation with Repair.

The last of the mutation operators, M3, selects an arbitrary point (not one of the original three) and attempts to alter its location by a bounded, random displacement. This operator does not change the number of points in the path by itself but additional points can be added when the alteration results in an infeasible path. When the bounds of the displacement are loose, the resulting path is more likely than not to be infeasible. Additionally, loosely-bounded displacement results in a greater number of points being added due to repair. On the other hand, if the bounds of the displacement are too tight then the operator becomes nothing more than a tool for local search. The path planning experiments given in Chapter 5 attempt to illustrate the tradeoffs associated with loosening or tightening the bounds of the alteration. Figure 3.8 illustrates the alteration mutation operator.

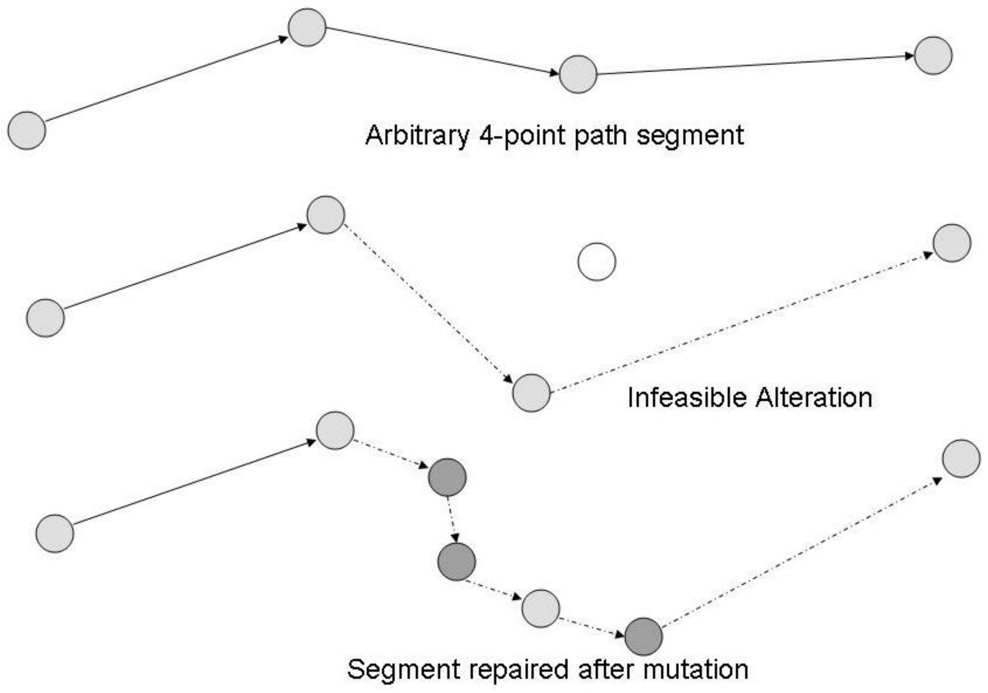

Arbitrary 4-point path segment
Infeasible Alteration
Segment repaired after mutation

Figure 3.8 Mutate Alter Operation with Repair.

3.1.2 Parallel Swarm Simulator. The swarm simulator routes individual members to required targets by way of required waypoints. These way points are generated by the path planner and assigned to sub swarms by the vehicle router. They are designed to minimize climbing, distance, and risk.

- Problem Domain. The problem of directing swarm behavior can be expressed as the cumulative problem of directing individual member behavior. The following relations mathematically define the problem domain of the swarm model:

Given a swarm member $\boldsymbol{v}_i$ and the following:

1. A terrain region (X, Y) with an elevation $Z = f(X, Y)$
2. A neighborhood vehicle set $\boldsymbol{V}$
3. A next waypoint w_{next} **< i, j, k >**
4. A current position s(t) **= < i, j, k >**
5. A set clearance C

Create a vector $\boldsymbol{v}(t+\Delta t)$ to guide $\boldsymbol{v}_i$ toward w_{nex} subject to conditions:

1. $z(t+\Delta t) \simeq C + f(x_{t+\Delta t}, y_{t+\Delta t})$
2. $|s(t+\Delta t) - w_{\text{next}}| < |s(t) - w_{\text{next}}|$
3. $\forall v \in V, v \neq v_i, |s(t) - b(v_i(t))| > |s(t+\Delta t) - b(v_i(t+\Delta t))|$

Where condition 1 maintains the required set clearance, condition 2 moves the vehicle toward the next steering point, and condition 3 adjusts the separation between the member $\boldsymbol{v}_i$ and all neighbors in $\boldsymbol{V}$ toward the proper separation distance b.

The AFIT UAV Swarm Mission Planning System combines the simple behavior rules of Kadrovach [13] [23] with the ability to achieve higher-level goals such as those described in [46]. While high-level goals such as reaching targets and path planning are left to the individual members in Reynolds, the AFIT system uses the router and planner to achieve them. Specific steering behaviors are defined in the swarm simulator tool. This approach simplifies the simulation model which increases its scalability. This section gives the behavioral description of the simulation model and its high-level design.

3.5.1 Simulation Overview. The swarm simulator is given a set of preplanned swarm routes that have been optimized by the planner and the router along with a digital terrain data set for the swarm's area of operation. The model calculates a set of trajectories for every swarm member. These trajectories embody the planned paths to targets as well as the application of the behavior rules for the members. The model output is then visualized using the Sky View visualization package.

3.5.2 Behavior Model. The behavior model consists of a set of rules to achieve path-following swarm behavior, a set of modes under which the rules are applied with various weighting factors, and a neighborhood of influence which defines which members affect the behavior of a given member. Each rule results in a unit vector addition operation applied to an individual. The sum of these vectors produces the member's trajectory.

-Neighborhood. Just as with swarms of insects or flocks of birds, swarms of UAVs have limitations on information that can be obtained from other members of the swarm. These restrictions are generally based on the proximity of a member to other members of the swarm. In the model presented here, we define the notion of neighborhood which is used to define the communication model as well as shape of the swarm formation.

The swarm shape is a 3-D stack of diamond tessellations. Each plane or level in the stack is offset one half-step from the level directly above or below it. Figure 3.9 gives an example of a swarm with a width of five vehicles and depicts the head-on view.

Co-planar members are same-colored. The same figure also represents the top down view of the swarm. From that perspective, the red members represent the middle vertical level while the green members represent the top and bottom levels. The top level obscures the view of the bottom level.

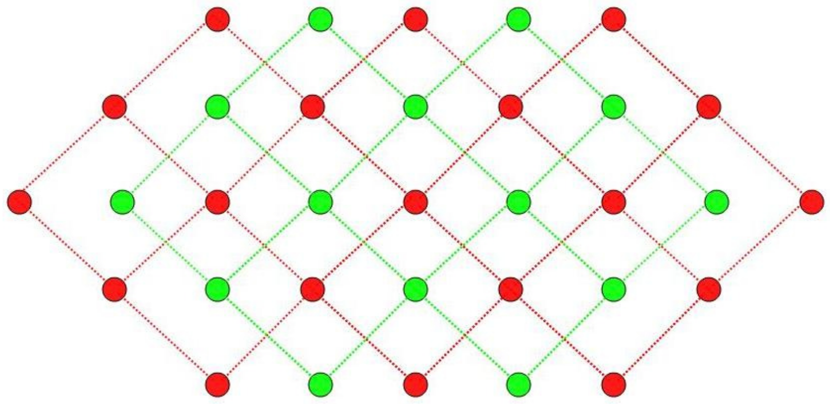


Figure 3.9 Head-On View of Swarm Formation.

The main parameter of the swarm formation is the separation *b,* representing the lateral distance between co-planar members and the distance of the co-planar neighbor directly in front and behind the member. Co-planar members 45 degrees front-left and front-right are at a distance of $b/\sqrt{2}$. Figure 3.10 illustrates the separation parameter.

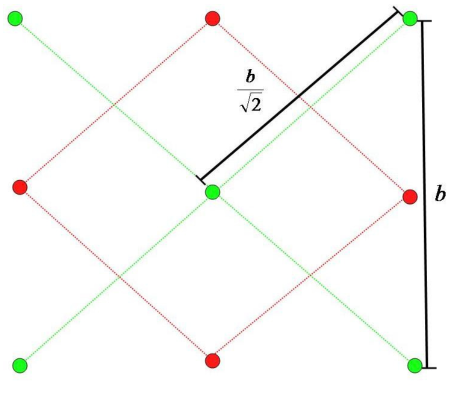


Figure 3.10 Swarm Separation.

Members are not influenced by those behind them for two reasons. First, the lead members are first to climb in response to terrain and also reach their target and begin their turns before trailing members. Application of the cohesion rule would cause lead members to throttle back when climbing or turning to allow trailing members to catch up. Instead, catching up is achieved by trailing members applying the cohesion rule with respect to their distance from the leading vehicles. A second reason for this simplification is a reduction of the communication overhead. Restricting the neighbor hood of a member to those members level with or in front of the member, reduces the size of the neighborhood considerably. Table 3.1 defines the neighborhood of influence surrounding a given swarm member.

Plane	**Distance**	**# Neighbors**
Co-planer	b	3
Co-planar	$b/\sqrt{2}$	2
Plane Above	$b/\sqrt{2}$	3
Plane Below	$b/\sqrt{2}$	3
Two Levels Up	b	1
Two Levels Down	b	1
TOTAL		**13**

Table 3.1 Neighborhood Definition.

- *Rules.* The behavior model consists of a set of three rules **R** = {r_1, r_2, r_3}. The application of these rule result from the interaction of individual swarm members with one another and with the terrain. As defined by Kadrovach and implemented by Corner in [9], each swarm member can only detect and be influenced by its neighbors.

r1 Cohesion. This rule is based on Reynolds' original model [45]. It creates a vector that causes a vehicle to move toward its neighbor whenever the distance to that neighbor exceeds the threshold distance value. Recall that vehicles in the lead with respect to the next target are not influenced by the cohesion rule except by their co-planar members to the left and right. Figure 3.11 illustrates the cohesion rule and its net effect on the member.

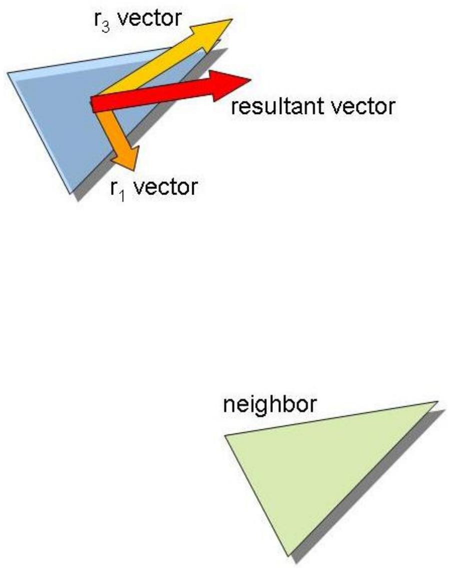


Figure 3.11 Cohesion Rule Applied to Swarm Member.

r2 Separation. Also from Reynolds, this rule adds a vector to the member moving it away from a neighbor when the distance to that neighbor decreases to below the threshold value. Leading vehicles have no members in front of them and are not directly influenced by those behind them. Therefore the separation rule applies only to their left and right co-planar neighbors and their neighbors two planes directly above and below them. Figure 3.12 illustrates the separation rule and its net effect on the member.

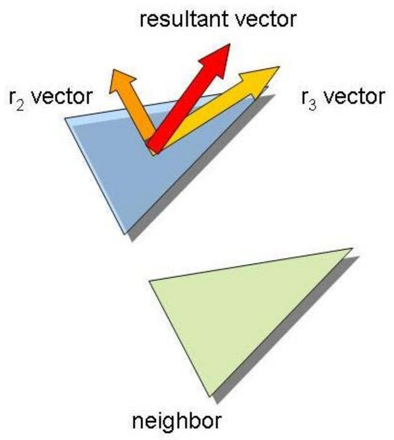


Figure 3.12 Separation Rule Applied to Swarm Member.

r3 Target Seeking. This rule replaces the more general alignment rule used in [9], [13], [23], and [45]. Figure 3.13 shows the application of the target seeking rule.

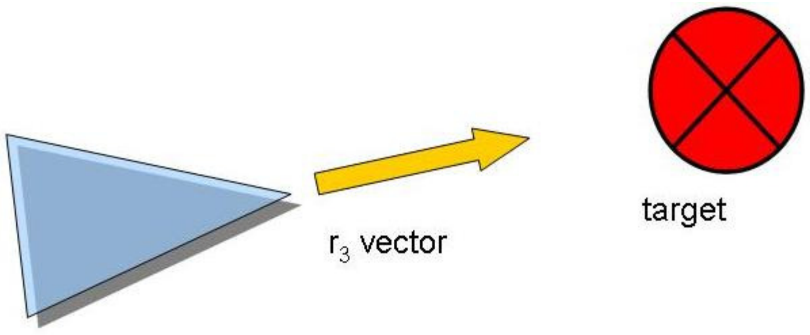


Figure 3.13 The Target Seeking Rule.

- *Modes.* The simulation progresses under two primary modes: warp and synchronization. During warp mode, communication among swarm members is suspended. Individual members continue on their path at their current heading. When small changes in individual trajectories are needed to avoid terrain, the other members are not notified. An individual member simply adjusts its trajectory as needed. During synchronization mode, members determine their neighborhoods and adjust their trajectories according to rules 1 and 2. The simulation enters synchronization mode under two conditions: 1) whenever a member alters its angular velocity by an amount greater than π/8 degrees, and 2) at scheduled fixed time intervals. The later condition is required to prevent drift in the swarm which would occur if minor changes in trajectory are extrapolated over long periods of time. During warp mode, the members apply only rule 3 which accounts for climbing and descending. Under synchronization mode, the swarm applies rules 1 and 2 with a weight of 20% and it applies rule 3 with a weight of 30%. This weighting was established empirically as optimal for maintaining swarm characteristics while achieving the target seeking behavior.

- *Communication Model.* The simulation is built on the SPEEDES framework as discussed in chapter 2. Under the original simulation, all UAVs in the scenario subscribed to all other UAVs. To simulate communication, influence was restricted to neighbors but messages were sent and received by all. In the current implementation, message traffic is restricted to neighbors and to the central simulation engine. This allows for true scalability of the swarm model. Details on the implementation of the communications neighborhood are given in Chapter 4.

- *Sub-Swarms.* The assignment of targets to sub-swarms is planned prior to execution of the simulation. Each vehicle therefore, knows which targets it is responsible for reaching. Since the entire swarm embarks on the mission from a single location, a swarm split is performed as sub-swarms go out in search of their individual targets. In order to minimize maneuvering and communication required for a split operation, the swarm uses a train or sausage link model in its original formation. Upon reaching a designated split point, the leading section of the swarm becomes a sub-swarm and turns towards its next target. The remainder of the swarm turns toward its next target. The split is done along the length of the swarm like a section of railroad cars being removed from the track. This method has the advantages of maintaining the shape of the sub-swarm and reducing the swarm's spatial footprint over time. Figure 3.14 illustrates the split operation on a swarm.

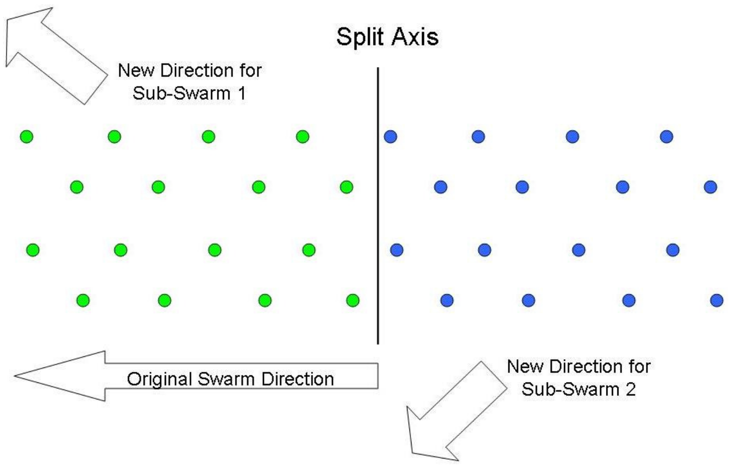


Figure 3.14 Swarm Split Operation.

Once a swarm has split, there is no join operation defined. At the end of the mission, all swarms return to their embarkation point. Due to varying target assignments, the sub-swarms return home separately.

3.2 System Level Design

The previous section addressed the design objectives of the UAV Swarm Mission Planning System and the problem domain/algorithm domain definitions of the three problem components. This section addresses system level design goals and integration. First, the system's data flow is defined illustrating the interaction of the design modules. This is followed by a discussion of the required interface between the modules and the integration of the problem domains. Parallelization of the path planner is then presented followed by the modification requirements for the router.

The system's data flow begins with creation of a target set, terrain field, threat lay-down, set clearance, and number of available swarm vehicles. The terrain masking algorithm is given the terrain elevation data, location and range of threats, and the set clearance or above ground altitude at which the vehicles fly. The threat lay-down is superimposed over the terrain grid, and grid areas considered to be within the effective detection and kill ranges of the threat are identified. The algorithm then calculates the line of sight visibility of each grid space within the effective range. An individual grid space is eliminated from the effective range of the threat when a terrain barrier lies between the grid space and the threat such that a line drawn from the threat radar to the grid point intersects the terrain boundary thus obscuring the grid space from sight of the radar. The set clearance of the UAV is added to the elevation of the grid space to account for the vehicles height above the ground. The updated threat range data is then stored for use by the path planner.

Once the terrain has been preprocessed, the vehicle router optimizes the assignment of vehicles to targets. To accomplish this, the router needs to know the complete cost associated with a particular route. The router produces a set of candidate solutions and invokes the parallel path planner to provide complete, feasible paths for each route. The router's genetic algorithm finds the lowest cost vehicle assignment for the mission, and retrieves the complete set of waypoints for each vehicle or sub-swarm. This complete set of paths is then fed to the parallel swarm simulator which then simulates the mission and produces a visualization of the swarm flying its mission. Figure 3.15 illustrates the dataflow design of the integrated system.

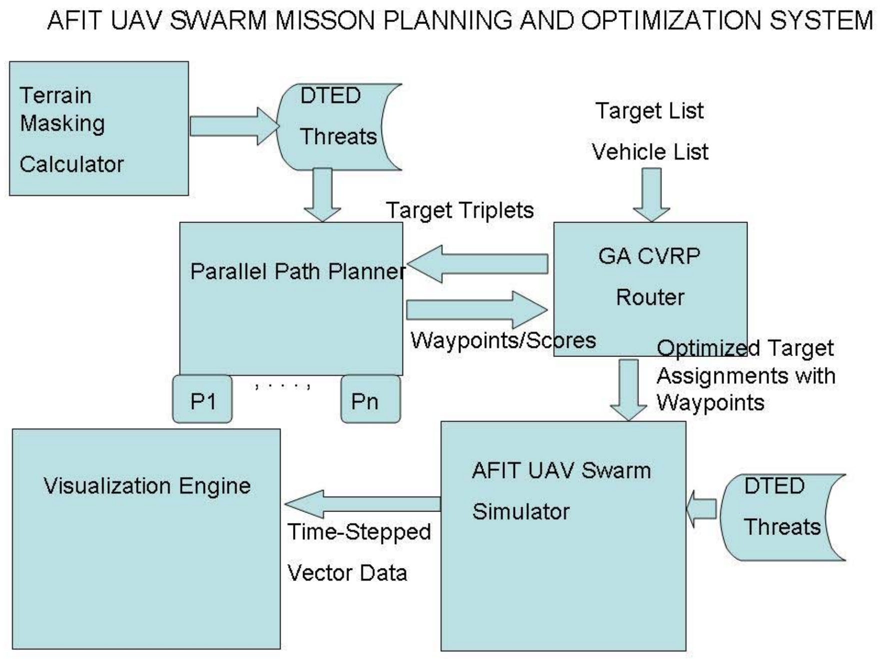


Figure 3.15 High Level Design of AFIT UAV Mission Planning System.

The path planner produces a solution to the problem of minimizing the risk and cost associated with moving a vehicle from one location to another by way of an intermediate point. The input to the algorithm therefore, is a triple $\{P_i, P_m, P_f\}$. The output contains the set of waypoints between P_i and P_m and between P_m and P_f. This output forms a single segment of a solution to the larger vehicle routing problem which contains multiple targets and multiple vehicles.

The router creates permutations of locations representing an ordered set of assignments to a set of vehicles. These permutations require sets of the triples described above. The generation of these triplets is time consuming and the number of possible triplets grows at a rate of $O(n^3)$ with the number of locations n. This growth rate is mitigated by three methods. First, the path planner is parallelized so that several paths can be generated at once. Next, the router uses a simple set of heuristics to request paths before they are needed. Finally, links which have already been calculated are cached for later reuse.

3.2.2 Integrating Problem Domains of the CVRP and the Path Planning Problem. In the CVRP, nodes in the graph represent targets or locations that must be visited and "serviced" by a vehicle. The set of locations or targets in the CVRP are the inputs to the path planner which creates the actual flight paths for the vehicle.

Each candidate solution in the genetic routing algorithm consists of a set of target links for each vehicle. Using Genetic Vehicle Representation, each solution in a population has the form $\{n_1,\ldots n_n\}$. This form efficiently encodes an entire set of nodes (targets) to the available vehicle. Decoding the route requires knowledge of each vehicle's range and each node's required payload. In this research, all vehicles are considered homogeneous and therefore have the same capacity. Once decoded, each vehicle's route is translated into overlapping triplets. For example, a vehicle given the assignment $\{depot, n_1, n_2, n_3, depot\}$ is translated into the following set of ordered triplets: $\{(depot, n_1, n_2),(n_1, n_2, n_3),(n_2, n_3, depot)\}$. These triplets, which replace links or arcs of the traditional TSP, are reused many times throughout execution of the routing algorithm. The router has been modified with a data structure to cache previously-requested paths. Once a single link is calculated, it is then archived for use in subsequent solutions. During the evaluation of a single candidate, all needed links are identified and the path planner is invoked to compute the solutions. The parallel path planner consists of a master node that receives the requests from the router, and *n* computing nodes, to calculate up to *n* paths at a time. The figure 3.16 describes how the router scores individual solutions, requests needed solutions, and caches solutions once they are created.

Algorithm: High-Level Interface between Planner and Router

```
1: procedure GVRGenome::EvaluateWithRepairs($\boldsymbol{L}$, $n$, $\boldsymbol{P}[start][mid][end]$)
2:     cost of route = 0, $\boldsymbol{L}$={ }
3:     for $i$ in 1 to $n$ do
4:         if $\boldsymbol{P}[start_i][mid_i][end_i]$does not exist do
5:             $\boldsymbol{L} = \boldsymbol{L} + (start_i),(mid_i),(end_i)$
6:         end if
7:     end for
8:     Planner($\boldsymbol{L}$)
9:     for $i$ in 1 to $n$ do
10:        cost of route = cost of route + f($\boldsymbol{L}_i$)
11:        $\boldsymbol{P}[start_i][mid_i][end_i]$ = f($\boldsymbol{L}_i$)
12:    end for
13: return cost of route
14: end procedure
```

Figure 3.16 Interfacing the Planner and Router Modules.

When fewer than *n* paths are requested, the remaining nodes are idle. When more than *n* paths are required, the first *n* nodes are assigned problems to solve. The first node to reach a solution then requests the next problem from the master node until all paths have been produced and returned to the router. A review of previous tests using the

Linn-Kerigan Travelling Salesman Problem (TSP) benchmarks revealed that the router has a 99.5% reuse rate in links requested. As the router's evolutionary timeline proceeds, the hit rate of the archive naturally increases. A consequence of this fact is that fewer routes are sent to the parallel planner as the evolution matures resulting in ever-increasing wasted computation time of the planner's nodes. To compensate for this idle time, the router uses three heuristics of the problem domain to request paths that are likely to be used later.

In the TSP problem, all routes begin at the depot or launch point. Therefore, routes containing the depot location as its first node are more likely to be found in a solution. Further, all solutions of the TSP require routes to end at the depot location or return point. Therefore, routes ending at the depot location are more likely to be contained in a candidate solution. Finally, if all route segments that either begin or end at the depot have been created, then arbitrary routes are requested to fill any remaining nodes. Once the needed paths of the evaluation function have been identified, remaining processors of the path planner are given requests to fill any idle processor slots using the following logic:

Algorithm: Requesting Extra Routes from the Parallel Path Planner

```
1: procedure REQUEST ROUTES(n, m, p, P[start][mid][end], L)
2:     n = number of paths requested in candidate solution
3:     m = p-n
4:     while m > 0 do
5:         if P [depot][rand][rand] does not exist then
6:             L = L + {depot, rand, rand}
7:             m = m-1
8:         else if  P [rand][rand][depot] does not exist then
9:         L = L + {rand, rand, depot}
```

```
            m = m-1
        **else if *P* [rand][rand][depot] does not exist then**
            L = L + {rand, rand, rand}
            m = m-1
        **end if**
    **end while**
    call PLANNER(***L***)
**end procedure**
```

Figure 3.17 Requesting Additional Routes from the Path Planner.

3.3.3 Parallelization of the Path Planner. A key element of choosing a work partitioning scheme is whether the algorithm is dynamic (the work load varies as the algorithm is run) or static (fixed amount of work, known a priori.) Using the benchmark TSP problem set as a guide, the cache hit rate is over 98%. However, during the first solution evaluation of the first generation, no solutions exist resulting in a very large miss rate and correspondingly large computational requirement. Empirical tests show that typical benchmarks result in about 40-50 new triplet requests in the first chromosome evaluated. As the algorithm progresses, more and more solutions are cached. This results in as few as zero triplets requested in the evaluation of a particular chromosome. Clearly the parallelization of the workload is a dynamic problem.

Several methods of partitioning dynamic workloads are presented in [21]. Asynchronous round robin (ASR) is a method where each compute node attempts to get more work from a neighboring processor corresponding to a variable *target*. The value of *target* is incremented modulo the number of processors available *p*. The method terminates when all processors run out of work.

Global Round Robin has only one target variable stored in a globally-accessible space. Each processor competes for this locked variable, increments it modulo p and attempts to get work from that processor. Often the contention for the shared variable creates a bottleneck of idle time when multiple processors are looking for work.

Random polling is a simple scheme in which an idle processor requests work from a random donor. Each processor is chosen with equal probability so work requests are divided evenly.

Although the workload associated with each call to the GVR evaluation function is variable from one call to the next, the workload of each job is fairly constant. Therefore, partitioning is most efficiently done at the chromosome level rather than within the path planner itself. This means that an efficient scheme is one that keeps each processor busy with as little overhead as possible. In this design, a master node reads the set of required jobs from a file created by the router. Up to *p* jobs are sent to the *p* available processors to process. As each job completes, the master node sends the next of the remaining *n-p* jobs to the processor that most recently completed a job. Once all jobs have been requested, any idle nodes are given a termination signal as a queue to shut down. Figure 3.18 details the algorithm for dividing the workload.

Algorithm: Parallelization of Path Planner

```
1: procedure PARALLEL_MOEA::EnvokePlanner($\mathcal{J}$, $p$ ,$n$)
2:   jobs remaining = | $\mathcal{J}$ |
3:   done = false
4:   jobs requested = 0
5:
6:       if $n > p$ then
7:           send $p$ non-blocking job requests to $p$ processors
8:           jobs requested = $p$
9:       else
```

```
            send $n$ jobs to $n$ processors
            jobs requested = 0
        while not done do
            Receive job output from processor $p_i$
            jobs remaining = jobs remaining -1
            if jobs requested = | $\mathcal{J}$ | then
                sendTermination( ) to $p_i$
            else
                send next job to $p_i$
                jobs requested = jobs requested + 1
            end if
            if jobs remaining = 0 then
                done = true
            end if
        end while
end procedure
```

Figure 3.18 Parallelization of the Path Planner.

- Enhancement of the GVR Router. The key modification of the router is made in the evaluation function. The original router created a static 2-D array of distances between every pair of targets in the problem instance. As each route is decoded and evaluated, the program simply reads the distance from the array and added that value to the overall distance of the route. The total 2-D Euclidian distance of all routes is the value used to determine fitness.

In this research, the 2-D array has been replaced with a 3-D cache of link weights representing the complete cost of each link triplet, $\{P_o, P_m, P_f\}$. Because only a small subset of all possible links are used in the Genetic Algorithm of the router, link weights are calculated by the parallel path planner only as needed. These values are then cached and reused.

3.3 Summary

This chapter presents the design objectives of the AFIT UAV Swarm Mission Planning System. The objectives are compartmentalized into three functional design modules: a multi-objective evolutionary algorithm-based parallel path planner, a genetic algorithm-based vehicle router, and a swarm behavior simulation model. A detailed strategy for system-level integration is defined. Chapter 4 presents the low–level design to include: the software development approach, choice of development tools, system architecture, data structures, methods, and the communication framework.

4. Low Level Design

This chapter discusses the low level design details of the AFIT UAV Swarm Mission Planning and Parallel Simulation System. Discussion begins with the software development approach used to create the individual system components followed by the approach used in the system-level integration. A detailed description of the data structures and methods used to implement the algorithms described in Chapter 3 is presented along with a description of the system-level user interface.

4.1 Software Development Approach

The software development effort consisted of four main thrusts: creation of the parallel path planner, modification of the existing router, modification of the simulation and visualization tools, and system level integration.

4.1.1 Software Development Approach of the Parallel Path Planner. The path planner is developed using an object-oriented (OO) approach and is written in C++. The path planner has a naturally hierarchical structure. For example, a population consists of a set of paths, and a path consists of a set of points. This structure lends itself to object encapsulation. Methods are defined that act on objects at various levels of abstraction. Where the approach used differs from the traditional OO approach is in the area of information hiding. Typically an OO design defines strict controls on the access to an object's data members. Specific methods to access or alter an object are used to govern the range within data must be assigned and to control which objects are authorized to act on other objects. The cost of this level of control is paid in terms of the runtime stack.

Every pair of accessor and mutator calls result in two additional functions and their associated parameters being loaded onto the stack. This cost is incompatible with the goals of this research. Therefore, the use of public data members is found throughout the developed code. This affords the benefit of modularity associated with object-oriented development without the associated overhead.

Parallelization of the path planner is developed using the Message Passing Interface (MPI). The primary reasons for choosing MPI are its compatibility with C++ and its availability on the AFIT parallel clusters. Java RMI was also available but its use would require an additional layer of interface between the planner and the network. MPI allows the developer to define the high level communication among program nodes but requires minimal interaction with or knowledge of the underlying communication architecture. This feature of MPI allows for greater portability of the path planner.

4.1.2 Software Development Approach of the Router. The router, created by Russell, is developed with principles similar to those used in the path planner. An object oriented framework is used to decompose the problem logically but public data members were used to reduce the overhead associated with information hiding.

Software reuse was another core concept in Russell's design. The genetic algorithm that powers the router was developed using the GALib toolkit which is freely available under a BSD-style license [27].

Modification of the router is performed with the goal of minimal alteration of the existing code. In the original router, a static 2-D data structure stored link weights for all possible links in the graph. This was replaced with a larger 3-D data structure which is populated by invoking the parallel path planner. No modifications were made to the underlying genetic algorithm within the router nor were any significant modifications made to the program's interface.

4.1.3 Software Development Approach of the Simulation and Visualization Tools. The Simulator is built on top of the SPEEDES parallel environment. A high-level object oriented approach is used in its development. Objects are defined for UAVs and targets, but further object-based decomposition methods are not used. The code is modified to add the additional functionality described in Section 3.5 while maintaining as much of the original program structure as possible. By maintaining the program's core constructs, future research can augment this work with minimal integration problems. The code is written and compiled under BSD C++.

4.1.4 Software Development: System Level Integration. The underlying goal of the system integration effort is modularity. The system consists of distinct components each with separate functionality. File I-O is the main form of communication among the components. This minimizes dependencies among the modules and allows for replacement of the system's components. For example, the target set and vehicle list are fed via file input to the router. The router in turn creates a file-based list of paths it requires. Any path planning algorithm could be used providing it could read the input data file. For example, a path planner for stealth aircraft or higher-altitude reconnaissance aircraft would be less reliant on TF capabilities and would require other optimization criteria such as radar cross section. Replacing the planner with another would require minimal effort. Specifically, the data interface would need to be modified to be compatible with the router. File I-O is significantly slower than using shared variables, but its use requires much fewer dependencies between the programs thus making the system more modular.

4.2 Parallel Path Planner

The path planner consists of a domain-specific genetic algorithm and a low-overhead parallel architecture that allows for moderate scalability in the size of the routing problem. The specific methods and data structures used in the planner are defined. The data structures of the path planner are contained in a hierarchy. The main program has a population object, which consists of a number of path objects, which consist of a number of point objects. A single terrain object is created in the main program. The terrain object stores the DTED data and threat location and range information. Pointers to the terrain object are passed to population and to the individual routes. Figure 4.1 depicts the basic class structure of the path planner.

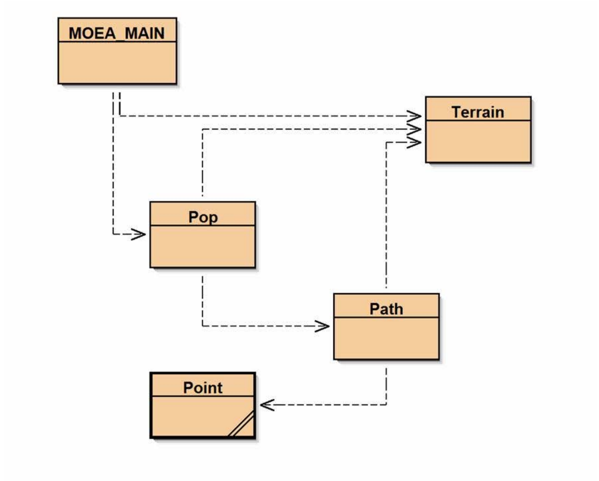


Figure 4.1 Class Diagram of the Path Planner.

4.2.2 Population. In the path planner, a population consists of an array of 50 path objects. The array data structure facilitates $\boldsymbol{O}(1)$ access to individual members of the population. Many of the evolutionary operators used in the algorithm iterate through the population so an array was a logical choice. In addition to the current population, a similar array structure is used to store the Pareto front archive. Once the population is initialized, all non-dominated members are copied and stored in the archive. With each evolutionary cycle, the current population is compared to the archive and dominated members are removed from the archive while new non-dominated solutions are added. Methods of the population include *Init(), Evaluate(), Evolve() Sort(),and Determine Dominance().*

- *Init Method.* During initialization, all 50 path objects of the population are created with the three point objects, start, middle, and end. Once the three point path has been created, a repair function is called to determine whether the heading change required in passing the midpoint is greater than 45 degrees. If it is, the repair function inserts points around the midpoint so that each turn is at most 45 degrees. This method uses the modified convex hull algorithm which is detailed in Appendix B. After repair, each path is then augmented with a set of random connecting waypoints. The random number generator used to select the location of the points called the Mersenne Twister. Details on the Mersenne Twister are found in [33]. The number of points added is proportional to the distances between the start and midpoint and between the mid and end point. Because of the 45 degree turn constraint, the area from which a random point is selected is restricted to ± 5 grid units in both X and Y directions from the midpoint of the line segment that connects the two points between which the new point is inserted. After each insertion is made, a check is done to ensure that the addition of the point does not violate the feasibility constraint. If a violation occurs, the point is removed from the path. Once the population has been initialized, the *Evaluate()* method is called on the population. Figure 4.2 shows the initialization of a single chromosome in three steps.

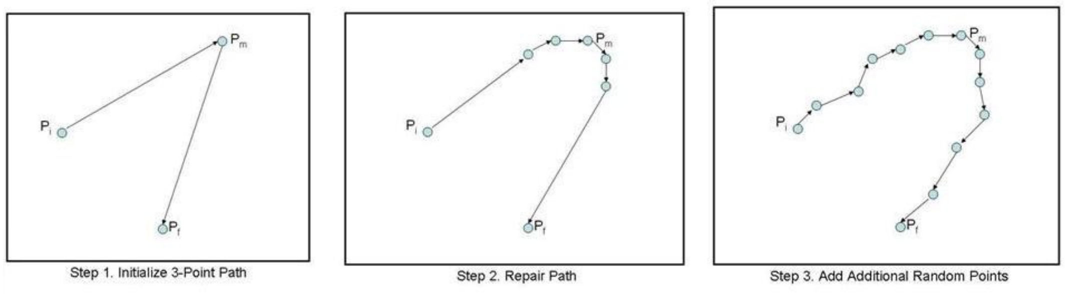

P_m
P_i
P_f
Step 1. Initialize 3-Point Path
P_m
P_i
P_f
Step 2. Repair Path
P_m
P_i
P_f
Step 3. Add Additional Random Points

Figure 4.2 Chromosome Initialization.

- *Evaluate Method.* The evaluate method is essentially a wrapper function that invokes the evaluation function of the path class which is described in section 4.2.3. Statistics for the population such as average cost score and average risk score are maintained in this method to give a window into the success of the evolutionary cycle. As the planner is further developed, these statistics can be used to adjust evolutionary parameters to speed up or slow down the rate of convergence. Once the population has been evaluated, it is sorted using the sort method.

- *Evolve Method.* Evolution consists of selection, crossover, and mutation operations. The selection operator chooses half the population for selection based on their rank. The population is grouped into quintiles. In the top quintile, nine of the ten members are randomly selected, In the second quintile, eight are chosen and so on until only two members are chosen from the bottom quintile. Once selected, the members are randomly selected for crossover. The crossover is performed as described in Figure 3.5. After crossover, each member of the next generation is mutated. The three mutation operators discussed in Section 3.1.

- *Sort Method*. There are several ways to sort the population which are controlled by passing the desired method as a parameter. The population is sorted by cost and then by risk to determine the dominance. It can also be sorted by the total score. Because of the limited population range, a trivial local serial sort algorithm is used.

- *Determine Dominance Method.* There are two main phases in this MOEA method. The first looks for non-dominated members in the current population. The dominance rank of the member is the main selection criterion. The second phase compares the new rank sorted population to the Pareto front archive to determine global dominance. In a *for* loop structure, each member of the population is compared to every other member by its risk and its cost scores. If existing Pareto members are now dominated by members of the current population, they are removed; if members of the current population are non-dominated with respect to Pareto front members, they are added to the archive.

4.2.3 Path Object. The path object is the chromosomal unit representing complete, individual solutions to the path planning problem. The path ADT consists of a linked list of point objects representing the flight plan, a set of statistical data including number of points and evaluation function scores, and a set of fixed data which includes the start, middle, and end points and a pointer to the terrain object. Figure 4.3 highlights the main data members of the Path ADT.

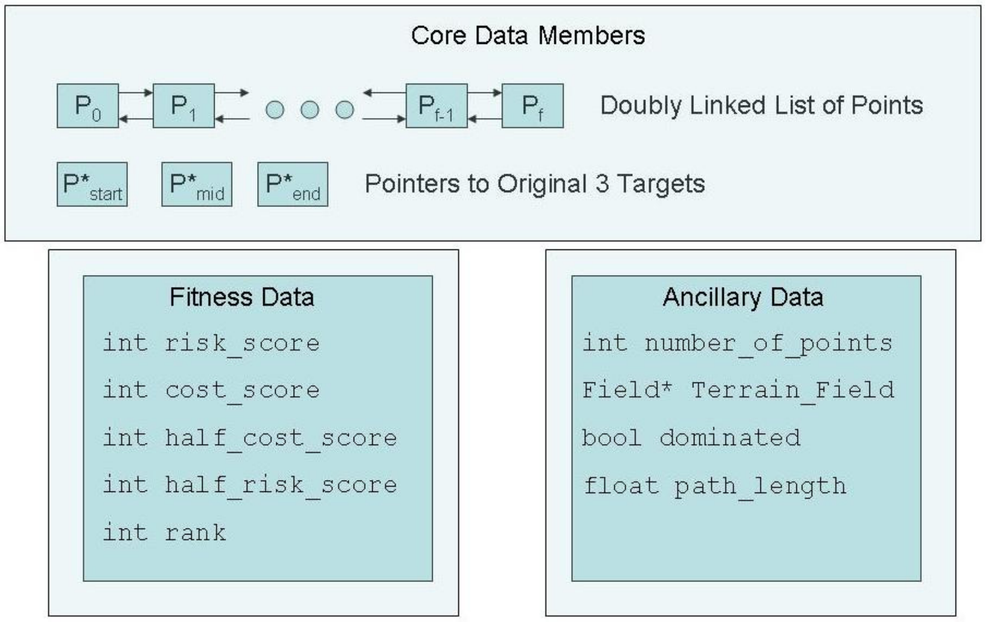


Figure 4.3 Data Members of the Path ADT.

Methods of the Path ADT include a constructor, copy constructor, add and delete point operations, point retrieval operators, repair operator, and a set of evaluation functions.

- *Constructor.* A path object is created with three point parameters representing the start, mid, and end points, a pointer to the terrain object, and an integer value, set clearance which is the AGL altitude the vehicle maintains during terrain following operations.

- *Add/Delete Point.* Given the linked structure of the path, the *add* and *delete* operators must iterate through the list of points to reach the index of insertion/deletion. A call to add a point with a value of 5 inserts the point after the 5th point in the list. Once the point is added, the heading is recalculated for the previous point and then calculated for the point inserted. A call to delete point with a value of 5 deletes the 5th point and adjusts the heading and pointer configuration of the 4th and 6th (now 5th) point. Because of the doubly linked structure of the path, only a single search for the insertion/deletion location is required to access all three points required to perform either operation. These operations are of course **O**(n).

- *Repair Operator.* When an insertion, deletion or alteration operation is done on the path, the feasibility constraint is often violated. If there is sufficient space between the points of the invalid segment, the repair function adds additional intermediate points to smooth the turns. The repair operator returns a value of *true* if the repair is successful. When *false* is returned, the calling module cancels the operation and returns the path to its previous state. Since the repair function is called only after the location of the point in question is known, no searching is required by this method. The number of operations needed to validate the heading changes between the points is constant **O**(c).

- *Evaluation Functions.* Two main evaluation functions are used to determine fitness of candidate solutions. The first function, cost, is a composite of the Euclidian Distance and the amount of climbing required for a path. The second, risk, is a composite of the hideability score, the linear distance of intersection between the path and a threat detection ring or kill ring.

- *Cost Function*. The distance portion of the cost function, simply starts at the path head and calculates the distance between the current point and the next until the end of the linked list of points is reached. In the climb function, the path is overlaid on the terrain grid. The sequential ordering of grid locations is mapped from the points in the route by means of the utility function *getNextBlock()*. The function then calculates the difference in elevation between the current grid point and the next. If there is an increase, the altitude difference is added to the cost. If there is a decrease, the score is unchanged. There are two primary reasons for this. First, since the vehicle returns to its starting location, there is zero net difference in elevation, thus the cost of climbing can be thought to include the eventual cost associated with descent. Further, climbing requires a great deal more energy (fuel) than descending.

- *Risk Function*. The intersection between the path and both threat and kill rings are determined exactly the same way as the climbing function. The costs of crossing through a single grid space in either a threat detection or kill zone is a parameter that is passed to the path planner at runtime. The hideability score is calculated by summing the number of visible points associated with each grid space the path intersects. These points are identified by the *getNextBlock()* function.1 The relative weight of hideability versus detection is also parameterized.

4.2.4 Point Object. The Point ADT is the allele of the path chromosome. Other than constructor/destructor operations, the point consists of only data members and pointers. Every point contains a pair of integer (*x,y*) coordinates, and integer altitude (feet MSL), an integer set clearance, and a heading. Pointers are maintained for next and previous points. The *set clearance* is the same as that of the path object, but it is maintained in the point object for future use, such as creating routes with different terrain following altitudes for different route segments. This capability is not implemented at present.

4.2.5 Interface to the Router Module. As described in Section 3.2, the router relies on the parallel path planner to determine the cost of each link or path in the route. These costs are stored in two 3-D integer arrays. The first array, `cost_matrix[n][n][n]`, stores the cost of the complete path $\{P_i, P_j, P_k\}$. The second array, `half_cost_matrix[n][n][n]`, stores the cost of the path only from the point P_j to P_k. This is needed because as the links are concatenated, there is an overlap between each link. This overlap is centered on the mid point and ensures that any 3-point link can be appended to any other 3-point link without violation of the turn constraint. Only the first link in the route, the segment which contains the depot or origin, is given the full cost of the link. The path planner returns both costs so the two arrays are maintained in parallel.

To maintain independence between the router and the path planner, the two modules use file I/O as their primary communication method. A complete set of path requests are written by the router and stored in a file, *linkFile*. The router then calls the planner with the *system* command. The planner reads the link file, generates all requests, writes them to a file, *pathFile*, and terminates. Termination of the path planner causes the *system* command to terminate which signals the router to continue. The router then reads the route scores from the path file, updates the two caches of scores, and completes the evaluation of the solution. Figure 4.4 expands the algorithm given in figure 3.2 and details the integration of the router with the planner

Algorithm: Integration of Planner with Router (Low Level)

```
1: procedure GVRGenome::EvaluateWithRepairs(R, cost_matrix[n][n][n],
            half_cost_matrix[n][n][n])
2:   int cost of route = 0
3:   for i in 1 to |R| do
4:       decode_route(int* route, int* links)
5:       for i in 0 to | links |do
6:           if cost_matrix[i][i+1][i+2] = 0 then
7:               write(linkFile, i,i+1,i+2)
8:           end if
9:       end for
10:  end for
11:  system(planner.exe, linkFile, pathFile);    //invoke planner
12:  open( pathfile )
13:  for i in 1 to pathFile.size do
14:      cost_matrix[pi][pj][pk] = $P_{i.cost}$
15:      half_cost_matrix[pi][pj][pk] = $P_{i.half\_cost}$
16:  end for
17: return cost of route
18: end procedure
```

Figure 4.4 Invoking the Path Planner from the Router.

Because genetic vehicle representation assumes routes beginning and ending at the depot location, the depot is not explicitly encoded in candidate solutions on the router. The path planner however requires explicit description of all three points required. The *DecodeRoute* procedure converts contiguous routes into triplets and ads the depot location to the beginning and end of the route. Figure 4.5 illustrates the *DecodeRoute* procedure.

Algorithm: Decode Route

```
1: procedure Utils::DecodeRoute(int* route, int* links)
2:     links[0]=0                    //preface route with depot location
3:     for i in 0 to |route| -1 do
4:         links[i]=route[i]
5:     end for
6:     links[|links|-1]=0            //append depot to end of list
7: end procedure
```

Figure 4.5 Decoding Routes into Triplets.

4.2.6 Parallelization. As described in chapter 3, parallelization is performed at the problem instance level. As the router requests multiple path links, the path planner is run in parallel to solve up to *p* requests at a time. The Message Passing Interface (MPI) is used as the development framework for the parallel communication. MPI is preferable to other available communication environments for two reasons. First, MPI is C/C++ based so it is easier to integrate with the planner versus Java RMI which is also available. Secondly, MPI is independent of the underlying physical architecture. It is therefore more portable from one parallel cluster to another than a custom-built communications model would be. Appendix C. describes the basic MPI constructs used in this design.

- *Parallel MPI Algorithm.* This section defines how the path planner is parallelized using MPI constructs. To illustrate the program flow, this description is given in the pseudocode of Figure 4.6.

```
// Initialize: MPI_Init(), MPI_Comm_rank()

int me; // node number

if me = 0 then // Code for Master Node

      MPI_Broadcast(); // Send Terrain Data
      MPI_Barrier();      // Wait for all to receive
      MPI_Braodcast();    // Send Threat Data
      MPI_Barrier();      // Wait for all to receive


      // load each processor with a job
      for i in 1 to p do
            MPI_Send(); //send single job to node
            MPI_Irecv();// non-blocking receive from slave node
      end for

      //Test for 1st job to complete
      while jobs_remaining > 0
            MPI_Test();
            if job has been received then
                  write output to file
            end if
            if more jobs to request
                 MPI_send();  //send next job to first available p
            else
                 MPI_Send();  //send termination signal to p
            end if
      end while

else // Code for slave nodes
      MPI_Bcast();      //broadcast receive of terrain
      MPI_Barrier();    //wait for others to receive
      MPI_Bcast();      //broadcast receive of threat data
      MPI_Barrier();    //wait for others to receive

      bool quit = false;
      while quit = false do
            MPI_Recv();       //receive first data request
            if termination signal received then
                  quit = true
            else
                  run path planning algorithm
                  MPI_Send(); //send results to master node
            end if
      end while
end if
```

Figure 4.6 Low-Level Parallelization of the Path Planner.

4.3 Swarm Simulation

This section discusses the parallel swarm simulation. First, a discussion of the simulation's underlying architecture, i.e. Synchronous Parallel Environment for Emulation and Discrete Event Simulation (SPEEDES) is presented. Next various modes of simulation execution are contrasted. A comprehensive list and description of attributes and methods is given followed by a description of the visualization system used to playback the simulation.

4.3.1 The SPEEDES simulation environment. SPEEDES is an open-source parallel discrete simulation framework developed in C++. Its primary purpose is to allow users to develop small and large optimistic time-managed simulations [57]. Parallelization of the simulation allows for simultaneous processing of events. Optimistic processing of events enhances performance by allowing some events to be processed out of order. Out of order execution avoids delaying received events scheduled at a future time, while waiting on the receipt of all events from earlier times. The AFIT UAV Swarm Mission Planning System uses the SPEEDES framework as its simulation engine. This section discusses the major elements of the SPEEDES framework used in this research.

- *Simulation Engine.* SPEEDES is a collection of C++ classes and an API that allows users to build simulations. Objects are the fundamental building block in a SPEEDES simulation. Simulations proceed as events are scheduled on objects. At the core of the simulation engine is a custom data structure called the Qheap. The Qheap manages lists of pending events and has been measured to be more than two times as efficient as traditional splay trees which are traditional event handling objects [57]. Additionally, the engine also has a fast internal hashing scheme that supports event cancellation and quickly associates events with objects.

- *Objects*. All state data representing an entity belong to a simulation object. Using an inheritance structure, all simulation objects are written to inherit from the class *SpSimObj*. This root class provides basic functionality such as the ability to schedule events, process event handlers and respond to interactions. Objects require methods that represent events taking place on the object. Once the object and its methods are defined, they are plugged into the simulation.

Objects are managed by object managers. One object manager is created for each type of object on each node in the simulation. Object managers create the objects on their nodes, initialize and clean up processes, perform dynamic object creation, manage subscriptions to interactions and event handlers, and manage subscriptions by external interfaces to object data.

- *Events*. Events act on simulation objects by altering their state information. Each event acts on a single object. The ordering of events is handled by a scheduling function. An object may call an event on itself, on another object or they may occur autonomously.

- *Event Handlers*. Events are often complex and require information about the world outside of an object to be meaningful. To this end, event handlers are developed for objects to schedule events on objects. They include a feature called a trigger which provides the ability to execute an event or not based on a parameterized set of Boolean conditions.

- *Proxies*. The ability of an object to learn of the existence of other simulation objects, to determine the node location of a particular simulation object, and to retrieve state information about another object comes from Proxies. Because SPEEDES simulations are run on multiple processors, proxies are vital for handling communication between objects on different nodes. In simulations occurring on a single node, shared variables could replace this functionality. At initialization, an object's attributes are read from a parameter file, *Objects.par*. A proxy maintains the objects updated attributes and manages the transmission of this information to the outside world.

- *External Interfaces*. SPEEDES simulations calculate state information for all simulation objects for the complete simulation time. In order to use this information, external interfaces are used to retrieve this global information for analysis and visualization. The SPEEDES API allows programmers to create external objects to collect simulation information. In this research, a Distributed Interactive Simulation (DIS) module is interfaced with the SPEEDES simulation to collect UAV vector information which is then visualized.

- *Breathing Time Warp Algorithm.* Discrete event simulations can proceed under one or more scheduling methods. These can be divided into two main categories: conservative and optimistic. Most simulations are subject to the causality constraint which ensures that events are processed in order with respect to time. There are cases where causality is relevant and there are those when it is not. For example, consider a scheduled pair of events: UAV 1 fires at target 1 at time t=3 and target 2 fires at UAV 1 at time t=2. Because UAV 1 is likely effected by the event at t=2, a simulation cannot process the events out of order. On the other hand, if the events were completely independent, the order would not matter.

A conservative simulation ensures that causality is always maintained by processing events in strict time-stamped order. This method guarantees the simulation behaves correctly with respect to time. A disadvantage is that possible benefits gained by parallel processing of events are often negated by having to wait for earlier events.

On the other side of the spectrum, optimistic simulations process events as they are received, with out regard to the causality constraint. This of course minimizes wait times and results in faster runtimes. The disadvantage here is that when dependent events are processed out of order, the simulation reaches an unrealistic state and produces incorrect results. To counter this effect, optimistic simulations require the ability to rollback state information to an earlier time whenever dependent events violate the causality constraint. Two early strategies for managing optimistic simulations were Time Warp and Breathing Time Buckets.

The Time Warp mechanism is based on the notion of global virtual time (GVT). In GVT, simulation time is synonymous with simulation time. It is purely optimistic, processing events as they are received without regard to causality [16]. If the past is changed and this change affects the current state, then the simulation is rolled back to the time before the violation occurred. Then, anti-messages are created to nullify future events that do not account for the change in the past.

In the Breathing Time Buckets algorithm [47], Processors handle events, exchange messages, perform rollbacks, and advance the simulation time in cycles. In each cycle, all events that do not precede one another are run. The minimum time of all events is called the global simulation time (GST). Each processor calculates its own approximate GST by noting the receive times of all messages sent by the events it processes [58].

Each one of these approaches has shortcomings. Time Warp is subject to constant rollbacks which can result in an unacceptable amount of anti-messages. Breathing Time Buckets may not be able to process enough events per cycle to remain efficient. A hybrid of these two algorithms, Breathing Time Warp (BTW), was developed in the Synchronous Parallel Environment for Emulation and Discrete-Event Simulation (SPEEDES) [58] and is used in the this research. The BTW algorithm is run in four distinct phases. In the first of these, the Time Warp Phase, the first N events processed beyond the GVT have their associated messages sent out. Next, the Breathing Time Buckets Phase processes events but does not release the messages. Instead, they are saved within the events. The event horizon is estimated using the minimum time tag of all unsent messages. Once the simulation proceeds through the event horizon, the third phase, GVT, is executed. During this phase, each node reads incoming messages and monitors whether the number of messages sent equals the number received. Once this condition is met globally, the GVT is set to local virtual time (LVT). In the final phase, all messages which have not been sent are sent synchronously from all events with time tags not greater than the GVT. Event lists are then cleaned up and the system returns to the Time Warp phase.

The swarm simulation uses a hybrid approach to event handling. Under normal operation, simulation time advances with all vehicles following their most recent trajectories. Adjustments in attitude are made in the vertical as each vehicle advances through the terrain. On a periodic time interval, the UAVs communicate with one another to determine the degree of rules violation developed since the last synchronization. During this time, the simulation can proceed in a conservative mode, i.e. wait until all messages are received from all neighbors, or it can attempt to move forward with only a partial message set. If a UAV advances, but requires information received later, the simulation rolls back. If however, missing messages are not needed once received, then the simulation does not roll back. BTW is used in the later case. The experiments in Chapter 5 contrast the cost savings in permitting optimistic processing of messages versus the time spent in rollback.

When any member executes a sudden, sharp terrain-induced climb, that member will trigger an automatic synchronization to ensure that the swarm maintains its configuration. The frequency at which the swarm enters the scheduled synchronous mode and the threshold at which UAV velocity changes trigger non-scheduled synchronous operation are developed empirically in Chapters 5 and 6.

- *Rollback Types*. As discussed earlier, the use of any kind of Time Warp requires the ability to revert to an earlier state whenever the local causality constrain is violated. SPEEDES provides rollbackable data types to accomplish this. Nearly all common C/C++ data types have rollbackable countertypes in SPEEDES. In this research, two rollback types: *RB_Int* and *RB_Double* are used to rollback prior position and velocity attributes of UAVs.

4.3.2 Simulation Objects and Behaviors. The primary object used in this simulation is the UAV which is called *S_UAV*. At initialization, the S_UAV properties are initialized as static data members and read from a parameter file, *Formation.par*. These properties include:

- *NumInitUavs* – The number of UAVs in the swarm at initialization

- *swarmHeight* – The number of vertical layers in the swarm formation

- *swarmWidth* – The width of the swarm in number of UAVs at the center

- *swarmAltitude* – The terrain following altitude (AGL) of the center swarm layer

- *swarmSeparation* – Distance between swarm members at 45 degree angles

- *simTimeInterval* – Time step for the simulation output

- *uavSpeed* – Cruising Speed of UAVs under normal conditions

- *percentPath<0-n>* - Allocation of swarm assets to each sub-swarm *0-n*

Once swarm members are assigned their initial targets, their individual attributes are initialized:

- *xCoordinate* – Longitudinal coordinate

- *yCoordinate* – Latitudinal coordinate

- *zCoordinate* – Altitude (MSL) in feet

- *thetaCoordinate* – Heading Angle

- *phiCoordinate* – Climb/descend angle

These individual variables are rollbackable, representing each UAV's current direction and position information. Additionally, each UAV maintains a copy of its route which is the set of target coordinates. In the simulation, target coordinates represent not only actual targets, but also minor waypoints. These points are generated by the path planner and passed to the simulator as input.

As described earlier, functions that act on simulation objects are mapped to SPEEDES event types. The following functions perform the bulk of operations on the UAV objects. The functions are named and they are mapped to events of the same name but prefaced with "UAV." For example, the function *vehicleMove()* is mapped to the event *UAVvehicleMove*. Descriptions of these functions are:

- *vehicleMove()* – This is the main function that drives the UAV's behavior. During normal (unsynchronized mode), it uses knowledge of the vehicle's distance to its next target to generate its next velocity vector. In synchronize mode, the function also uses position information of its neighbors to generate its next velocity vector.

In the synchronized mode, the function begins by determining its set of neighbors. First, all UAVs for which the vehicle has information are sorted by distance. Then, the function loops through the ordered neighbors and assigns them into one of the 13 nearest neighbor slots. Once a neighbor is assigned, its shadow is calculated to determine which nearby vehicles should be removed from consideration. Because the list of UAVs is sorted, shadowing never causes a closer UAV to be obscured by a more distant one. Once this process is complete, each neighbor is analyzed to determine how far off it is spatially from the vehicle. A unit vector is generated to show the required movement the vehicle needs to make to conform to the rule of separation. This unit vector is then multiplied by a weighting factor which is proportionate to the degree of rules violation. The vectors generated by the complete scan of neighbors are then added to produce a resultant vector called *vectAttract*. This vector is then given a weighting factor α and added to the target distance vector which is given a weight of $1-\alpha$. The resultant vector is then used to update the UAVs motion.

When operating in unsynchronized mode, the neighbor vectors are not calculated and the target distance vector is given the full weight, i.e. $\alpha=0$. The major methods of the simulation are:

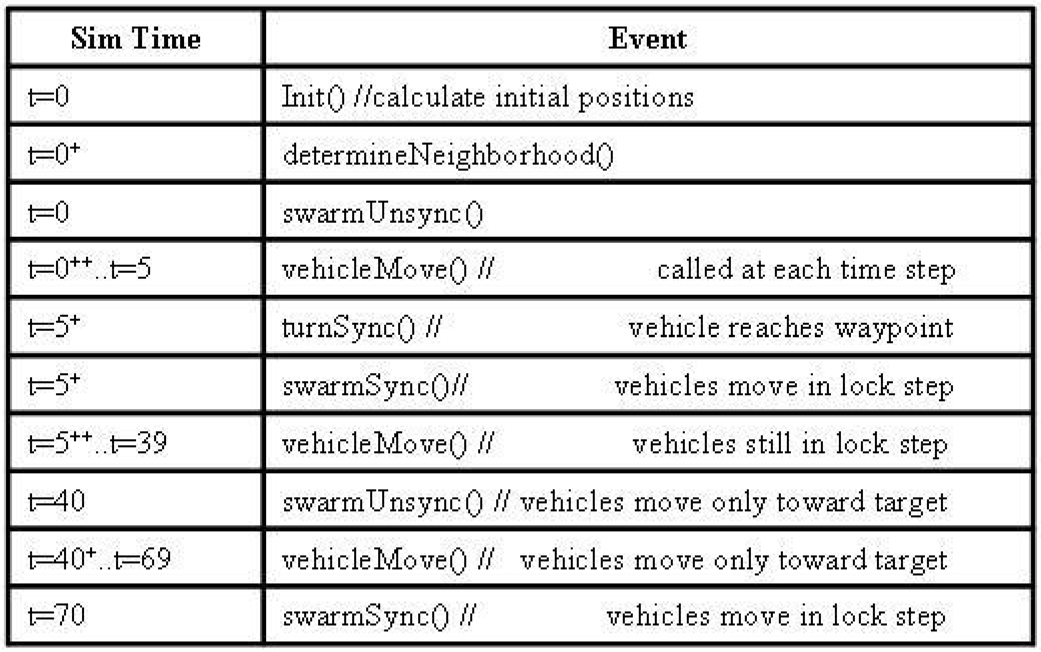
- *swarmSync()* – This function (event) causes the simulation to enter synchronous mode so that the swarm can adjust to maintain formation. It is scheduled periodically and whenever an individual makes a sharp change in direction. If the vehicle is engaged in a turn while this method is executing, the swarm uses its most recent neighborhood to adjust motion. If the vehicle is not engaged in a turn maneuver, then the members will re-calculate their neighborhood before adjusting their motion vectors.

-*turnSync()* – This event is scheduled on every UAV when the first swarm member reaches its next waypoint and lasts until the last member has cleared the waypoint. During this time, the neighborhoods are considered to be "fixed." Within each fixed neighborhood, the members synchronize at each simulation time step. This additional synchronization is designed to keep the swarm in formation during turn maneuvers. This event schedules a *swarmUnSync()* event when the last UAV has cleared the waypoint.

- *swarmUnSync()* – This event is scheduled several seconds after the swarm enters synchronized mode. It ensures that the vehicles use only the target distance vector to process their next movement vector.

- *removeUAV()* – When a UAV has completed its mission (or in future implementations, when the UAV has been destroyed), it is removed from the simulation. Members who subscribe to this particular UAV will no longer look for updates from this vehicle. Table 4.1 shows a typical execution segment highlighting events in the simulation.

Table 4.1 Typical Swarm Simulation Event Sequence.

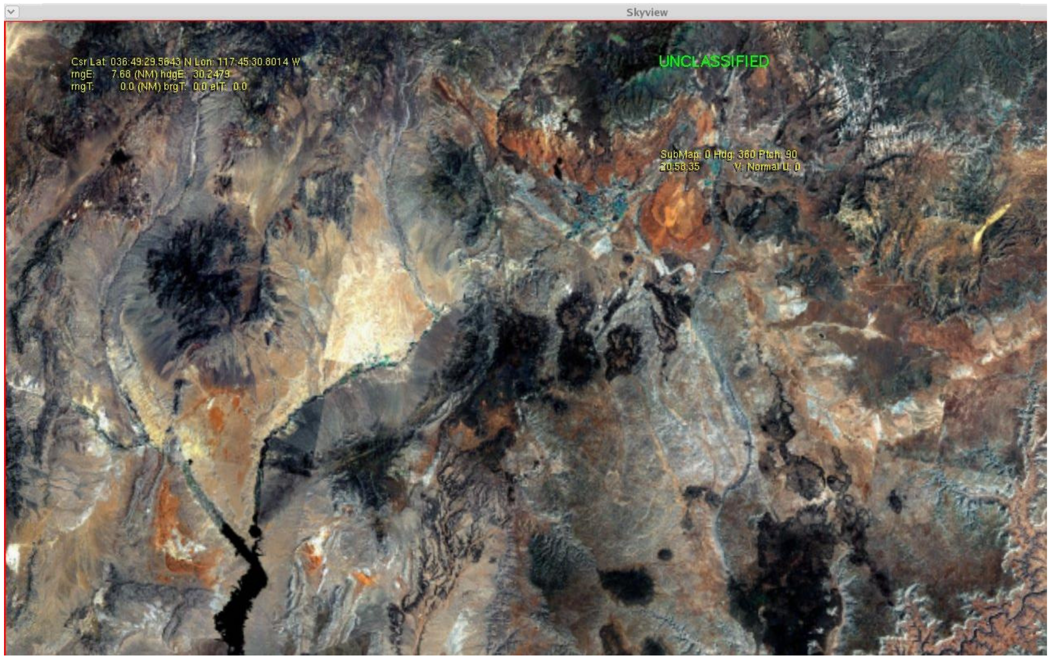


Sim Time	Event
t=0	Init() //calculate initial positions
t=0$^+$	determineNeighborhood()
t=0	swarmUnsync()
t=0^{++}..t=5	vehicleMove() // called at each time step
t=5$^+$	turnSync() // vehicle reaches waypoint
t=5$^+$	swarmSync()// vehicles move in lock step
t=5^{++}..t=39	vehicleMove() // vehicles still in lock step
t=40	swarmUnsync() // vehicles move only toward target
t=40$^+$..t=69	vehicleMove() // vehicles move only toward target
t=70	swarmSync() // vehicles move in lock step

4.3.3 Object Proxies and Communication. Because of the two main modes of operation, synchronized and unsynchronized, it is not necessary to continuously check for updates. This reduces the communication overhead considerably. Object proxies are queried only during execution of the *moveUpdate()* function and then only when operating under synchronized mode.

4.3.4 Visualization. The swarm model output is visualized offline after the simulation has completed. Two primary tools are used to provide this visualization: SkyView and the DIS Interface. These tools are the products of previously funded government research. They are commonly used in DoD research and are freely available within DoD.

-*SkyView*. Developed by the Georgia Tech Research Institute [20], SkyView is a three-dimensional terrain rendering program. It provides perspective views of areas of terrain based on satellite imagery and digital terrain elevation data. The software was developed using OpenGL and is freely distributed throughout DoD. The swarm simulation output is converted and visualized using SkyView. In this research, an operational battle space is created from DTED and imagery over Nevada in the vicinity of Nellis AFB. Figure 4.7 shows a SkyView rendering of this gaming area.



```
Csr Lat: 036:49:29.5643 N Lon: 117:45:30.8014 W
rngE:     7.68 (NM) hdgE:  30.2479
rngT:       0.0 (NM) brgT:  0.0 elT:  0.0
```

```
SubMap: 0 Hdg: 360 Ptch: 90
20:58:35          V: Normal U: 0
```

Figure 4.7 SkyView Terrain Visualization.

- *DIS Interface*. The SPEEDES simulation interfaces with an external module called the DIS Daemon. This module converts the model output, which consists of time-stepped UAV position and heading information, and transmits it to SkyView using the DIS communication framework. SkyView was specifically developed to interface with DIS. The module was developed by the sponsors of this research (Lt. Nick Amato, AFRL/SNZW). Using the raw position and heading information, the DIS Interface interpolates UAV attitudes and provides the ability to visualize UAVs climbing, descending, and turning.

4.4 Summary.

This chapter described the low-level design used in developing the AFIT UAV Swarm Mission Planning System. A detailed description of the methods and data structures used to implement the design of Chapter 3 is presented. Chapter 5 discusses the design of experiments used to validate the various design capabilities and limitations and to measure the system's performance characteristics.

5. Design of Experiments

This chapter describes the experiments conducted in this research. Recall from Section 1.3, that this research consists of three main components, a parallel path planner, vehicle router, and a UAV swarm simulator. The router is designed and tested in prior research [49] and is not extensively tested here. It is used primarily to input the problem instance into the overall simulation. It uses the parallel path planner to create individual route links which are then used to solve the CVRP. From there, the CVRP solution, complete with optimized paths, is given to the UAV swarm simulator where the swarm behavior model is applied to the problem and the results are visualized. Experiments conducted focus on two general areas: validation of the parallel path planner and effectiveness and efficiency of the swarm simulator. The testing approach is defined, as are the specific metrics used to gauge the system's overall performance.

5.1 Validation of the Path Planner

The primary purpose of the path planning algorithm is to effectively solve a particular form of the general path planning problem. Specifically, the path planner solves a three-target problem. The problem formulation is chosen to maintain compatibility with the Traveling Salesman Problem (TSP) and its variants (the VRP, CVRP, and CVRPTW). The experiments designed to validate the planner therefore focus on its effectiveness. Since the planner has several optimization criteria, the experiments focus on these individually.

5.1.1 Testing the Planner's Ability to Minimize Climbing. In this experiment, an artificial terrain field is created with a geometrically simple shape. The planner optimizes a route to the target by minimizing the climbing associated with the created path. Like all paths solved by the planner, this scenario consists of start, middle, and end locations. In between the straight-line path connecting the points are two large areas of high terrain which the planner must avoid. No threats are used in this experiment. Further, the weight associated with climb cost is maximized and the weight associated with distance is minimized to demonstrate the satisfaction of this single objective. Figure 5.1 shows the path to plan and the terrain.

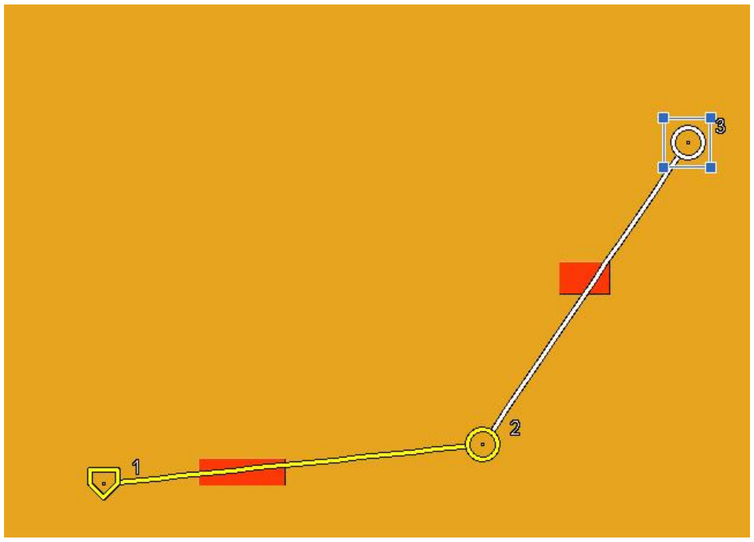


Figure 5.1 Terrain Avoidance Problem.

5.1.2 Illustrating Tradeoffs Between Cost and Risk. In this experiment, a real-world route is planned over Nevada in the vicinity of Nellis Air Force Base. The path planner is run first to minimize the cost of the route by minimizing distance and the amount of climbing associated with navigating the route. Then, the route is optimized to reduce risk. Hideability, the degree to which vehicles remain out of site of potential unknown threats is used as the optimization criterion. Figure 5.2 below shows the three-point route for the planner to solve overlain on a visualized DTED field.

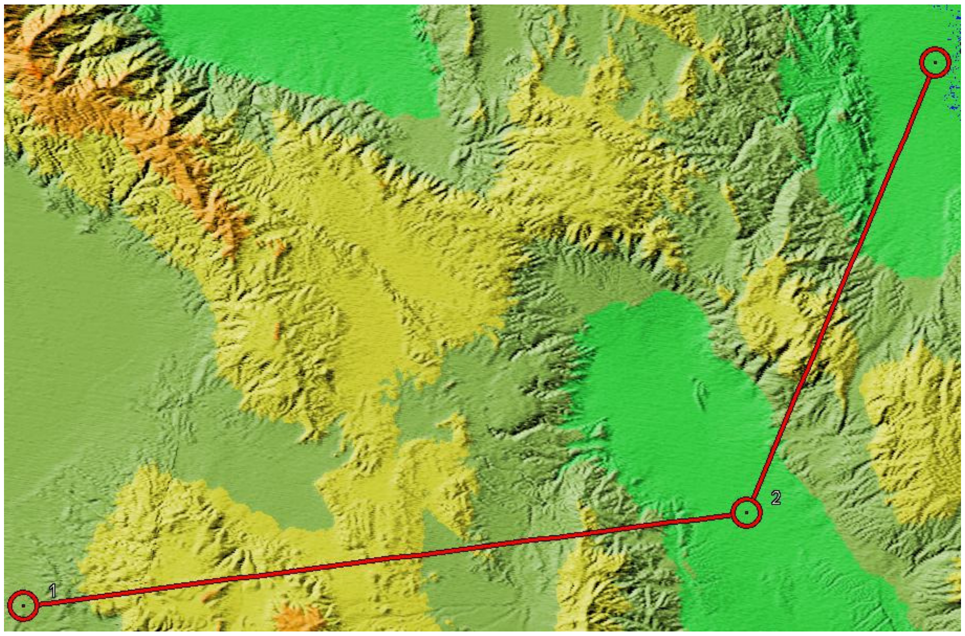


Figure 5.2 Nellis Route Over DTED Field.

5.1.3 Comparing the Planner to the Terrain Following Optimizer. This experiment compares the effectiveness of the path planner with AFRL's TFO algorithm. Three-element target packages are created along with a grid of real world terrain and a realistic threat lay down. The planner is run in single-objective mode and its solution is scored and recorded. The TFO is run and the output is passed into the evaluation function of the planner. The results are then compared. Because the TFO runs on a single processor Windows machine and the planner runs on a Linux cluster, no meaningful analysis of execution time can be made. Therefore, the cost function is the only means of comparison. The route used in this experiment is the same one as the previous experiment shown in Figure 5.2. To provide better geographic reference, the problem instance is shown on a CADRG navigation chart in figure 5.3.

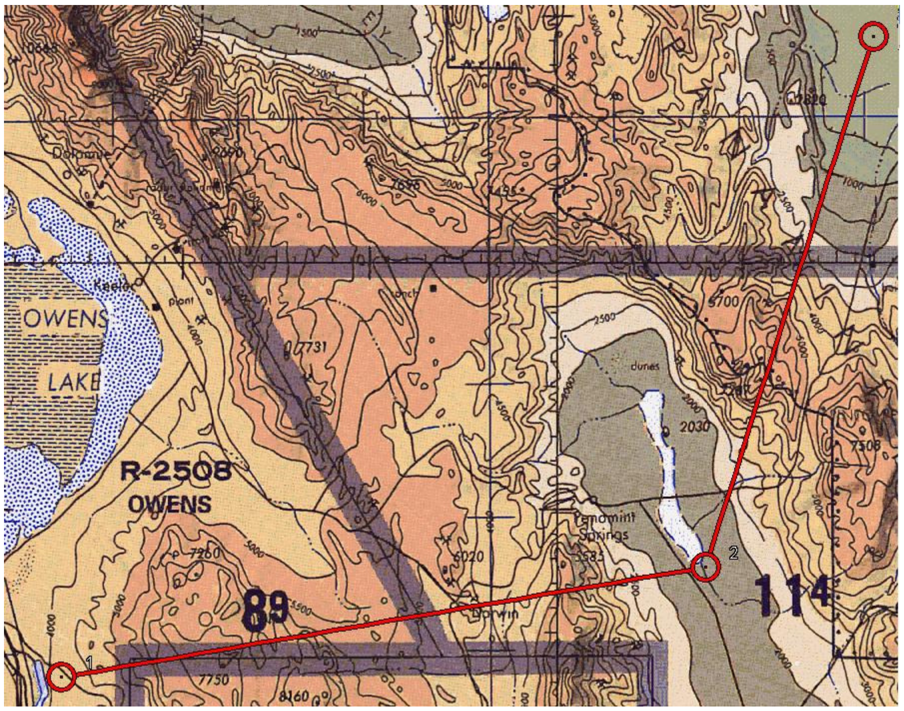
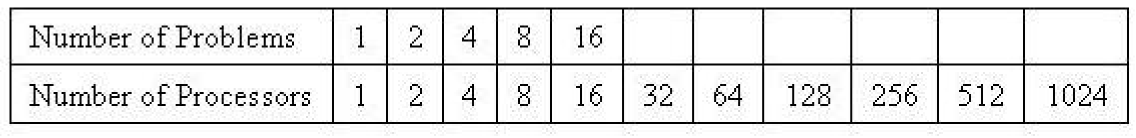


Figure 5.3 Nellis Route Over Navigation Chart.

5.1.4 Efficiency of the Parallel Path Planner. There are two general uses for the path planner. As a stand alone unit, the path planner is multi-objective, i.e. it provides decision makers with a range of solutions to a particular problem instance. The second use is to generate link solutions to the larger capacitated vehicle routing problem (CVRP). To measure the efficiency of the parallel path planner, the runtime of the serial version is compared with multiple-instance parallel runs of the algorithm. With this information, the scalability and speed-up of the algorithm is determined. The problem instance of section 5.1.3 is suitable for this experiment because the route: covers a wide range of the problem space, tests the repair function for a sharp turn, and is overlaid on a varied, real-world terrain space where Terrain Following Missions have been flown. Table 5.1 shows the parallel configuration of the test set conducted on AFIT's BANFF Beowulf cluster. Each test is run 30 times and the average runtime is computed.

Table 5.1 Parallel Problem Test Cases.



Number of Problems	1	2	4	8	16						
Number of Processors	1	2	4	8	16	32	64	128	256	512	1024

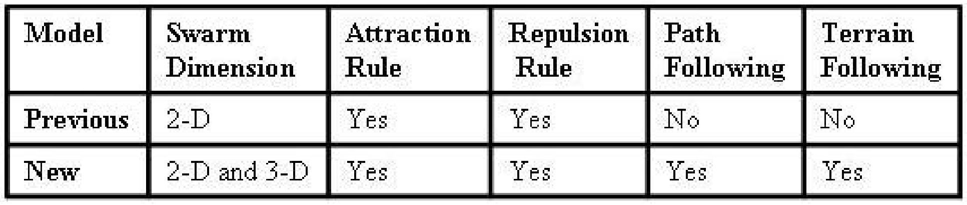
5.2 Integration of the Path Planner with the Router.

Modifications to the router only affected the data used by the routing algorithm. Naturally, scores from the path planner differ greatly from the static point-to-point scores originally used by the router. Experimentation in this area focus only on the ability of router to: successfully invoke the planner, make use of the planner's path scores, and complete its genetic routing algorithm.

5.3 Evaluating the Improved Swarm Simulator.

In the previous version of the swarm model, a 2-D swarm was placed into a uniform terrain region with targets or points of interest. The model demonstrated limited capability to find targets while conforming to the swarming rules discussed in chapter 2. In this section, a suite of experiments is developed to test the effects of the additional model capabilities on the swarm model. Table 5.2 illustrates the behavior enhancements of this research as they relate to the previous model. Testing focuses on adherence to the swarm rules as defined in Section 3.3 and the scalability of the enhanced model. The three major behavior enhancements are tested independently and collectively.

Table 5.2 Swarm Model Behavior Enhancements.



Model	Swarm Dimension	Attraction Rule	Repulsion Rule	Path Following	Terrain Following
Previous	2-D	Yes	Yes	No	No
New	2-D and 3-D	Yes	Yes	Yes	Yes

5.3.1 Validation of Adherence to Swarm Rules. Specific actions taken by the vehicles to reach targets, often conflict with the swarming rules. These experiments test the ability of the swarm to maintain its physical integrity while reaching all assigned targets in the route. Each of the following experiments use a common set of information to determine the adherence to the swarm rules. This information is derived offline from the simulation output file:

Neighborhood: A neighborhood, defined in Section 3.5.2, is calculated for each swarm member at each time step in the simulation output. Each neighbor has a required separation parameter based its position relative to the central UAV as defined in the parameter file.

Rules Threshold: To separate consequential rules violations from minor ones, a threshold violation level is set at 20% of the separation parameter. Rules violations in this experiment are then determined by instances when a vehicle's separation from any of its neighbors differs from its required distance by $\pm 20\%$.

Split Operation: The split operation defined in Section 3.4 is referenced throughout the remaining chapters. Figure 5.4 illustrates the split concept. It shows the top-down view of the swarm immediately after performing a split operation. The swarm size and sub-swarm assignment in the figure are used throughout the experiments.

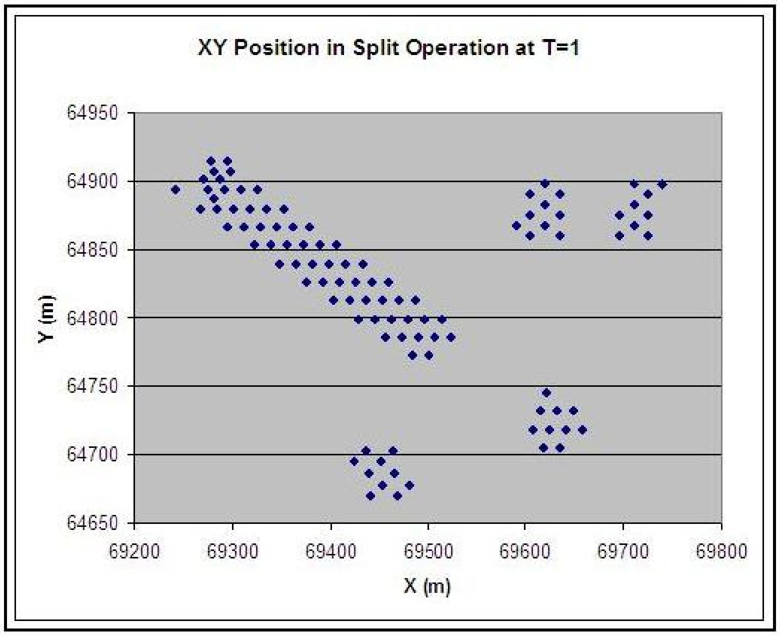


Figure 5.4 Top Down View of Swarm After Split Operation.

5.3.2 Swarm Rules Experiment Sets.

-Experiment Set 1. Objective: Observe the effect of the Path-following behavior on the swarm's cohesiveness.

- Metrics: From the simulation data, the average neighborhood size is calculated over time. The expected neighborhood size is defined in Table 3.1. Deterioration of neighborhood size is indicative of the swarm spreading out beyond its intended range.

-Method: The simulation is executed over flat terrain with only a single vertical layer. This configuration allows for isolation of the effects of path following from other model enhancements. For each time *t* in the simulation, the average neighborhood size is calculated using equation 5.1.

$$\bar{n}\,Size(t) = \frac{\sum_{t=0}^{f} |n|}{\#UAVs} \quad at \;\; time \;\; t \qquad (5.1)$$

Another measure of compliance is the degree to which rules are violated. Equation 5.1 measures how often the simulation is in a state of violation. To measure the degree of violation, the absolute value of each UAVs violation in meters is calculated at various time steps in the simulation. Equation 5.2 quantifies the magnitude of rules violations as:

$$for\, UAV\; i\, at\, time\, t : \frac{\left| \sum_{j=1}^{n} \left| vect\, diff\,(i, j) \right| - req.\; separation(i, j) \right|}{n} \qquad (5.2)$$

Where *vectdiff(i,j)* is the separation vector between the i^{th} and j^{th} vehicles and *req. separation* is the position-dependent separation distance required by the model's rules.

-Experiment Set 2. Using the same metrics of experiment set 1, the simulation adds terrain and terrain following behavior to the model. Any deterioration of cohesiveness observed between set 1 and set 2 illustrates the degree to which terrain following stresses swarm cohesion.

- Experiment Set 3. In this test, the swarm is configured in multiple vertical layers and the simulation is executed with the same path following and terrain properties in experiment 2. The addition of multiple vertical layers tests the 3-D swarm shape proposed in this research as a viable formation model.

-Experiment Set 4. In this set, the terrain is removed from the configuration used in Set 3. Differences in rules adherence between experiment sets 3 and 4 further enhance the validation of the swarm shape.

- *Experiment Set 5.* In addition to model behavior enhancements, the ability of the swarm to remain cohesive is influenced by the frequency at which the swarm synchronizes its movement. This experiment encompasses all three behavior enhancements of table 5.2 and measures the rules adherence as a function of synchronization period. Because Synchronization is costly, this test implicitly provides trade-offs between rules adherence and execution time. The experiment is conducted for synchronization periods of 30, 60, and 90 seconds.

5.3.3 Model Scalability Experiment.

-Objective: This experiment measures the scalability of the simulation and determines the speedup associated with adding additional parallel processors.

-Method: To look at the efficiency and scalability of the simulator, a suite of experiments are done to compare the run times of simulation with different sized swarms, varying number of CPUs, and varying swarm synchronization intervals.

-Metric: Speed-up is calculated as a means of determining the effect of greater parallelization on runtime. Table 5.3 gives the various parallel configurations and problem sizes used in this experiment.

Table 5.3 Parameters of Model Scalability Experiment.

UAVs	40	80	160	320	640
CPUs	4	8	16		

5.4 Summary. This chapter defined a comprehensive set of experiments designed to test the objectives of this research as defined in Section 1.3. Each of the three system-level components is tested individually with particular emphasis on the swarm model. Additionally, visualizations are conducted with simulation data that provide intuitive experiment validation criteria. Chapter 6 discusses the results of these experiments.

6. Results

This Chapter presents the results of experiments defined in Chapter 5. Each section presents the data and graphical results of the corresponding experiment in Chapter 5. Interpretations of the results are presented with emphasis on causal relationships between specific model enhancements and the observed behavior.

6.1 Validation of the Path Planner

6.1.1 Testing the Planner's Ability to Minimize Climbing. In this experiment, an artificial terrain field is created with a geometrically simple shape. Figure 6.1 below shows the optimized route from Figure 5.1 and the resulting terrain avoidance behavior.

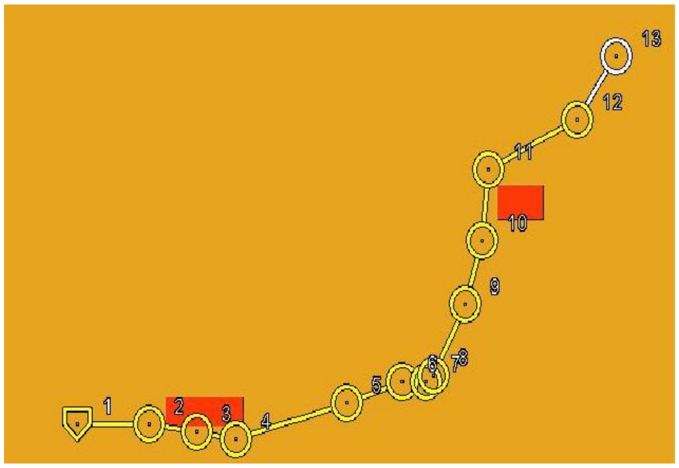


Figure 6.1 Path Planner Navigating Simple Terrain.

The light (orange) area represents flat terrain at 100ft (MSL). The darker (red) areas are flat terrain regions at 1100ft (MSL). Note that planner-generated route avoids the high terrain to eliminate climbing.

6.1.2 Illustrating Tradeoffs between Cost and Risk. Like the previous experiment, this one uses no threats in the scenario. Risk avoidance is measured in terms of hideability i.e. the ability of the planner to create routes that minimize the vehicle's visibility from surrounding locations. The planner was run in multi-objective mode and the least cost and least risk solutions were captured and visualized. Figure 6.2 shows the route of figure 5.2 optimized for cost minimization.

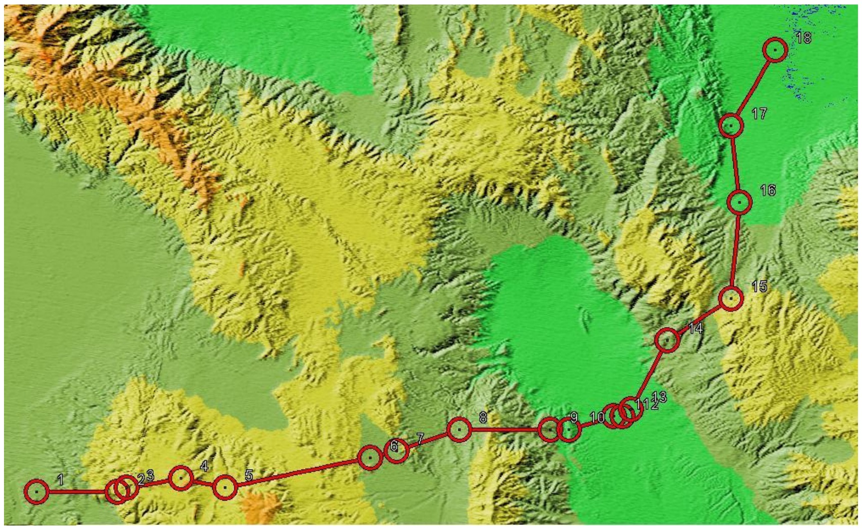


Figure 6.2 Route Optimized for Cost.

The route in figure 6.2 was scored according to the fitness functions. Its component scores are given in Table 6.1. From the same run, the lowest risk solution is given below. These two solutions represent the two extremes of the Pareto Front. Figure 6.3 shows a visualization of the lowest risk score.

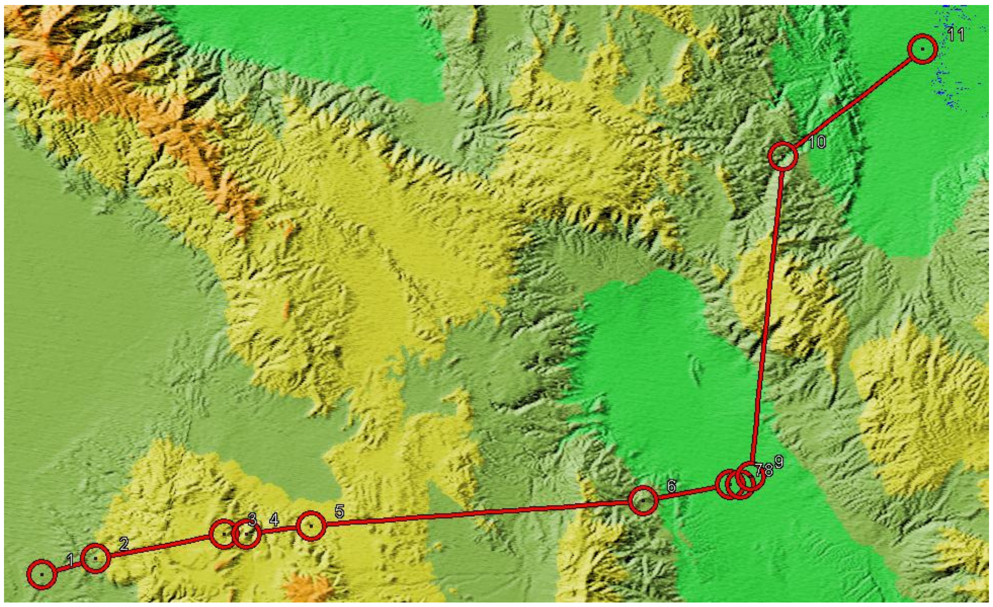


Figure 6.3 Optimized Low Risk Solution.

Table 6.1 Fitness Function Component Scores for Nellis Route.

Route	Path Length	Climb Cost	Hideability
Low Cost	94289	10522	14774
Low Risk	93857	12444	17904

6.*1.3 Comparing the Planner to the Terrain Following Optimizer.* This experiment compares the effectiveness of the path planner with AFRL's TFO algorithm. This experiment was performed by running the problem instance Nellis Route 1 on the path planner and comparing the results with TFO's solution. It should be noted that TFO was not able to solve the problem directly. Due to algorithmic constraints of TFO's tree search, a maximum of 20nm are allowed between targets. As a result, intermediate points had to be inserted between the targets before the route could be optimized. An additional limitation of TFO is that it optimizes paths between targets but does not optimize connections between targets. Therefore, TFO does not allow more than 45 degrees of heading change between consecutive major waypoints. The parallel path planner has neither of these constraints. Figure 6.4 depicts the TFO solution to the problem instance Nellis Route 1.

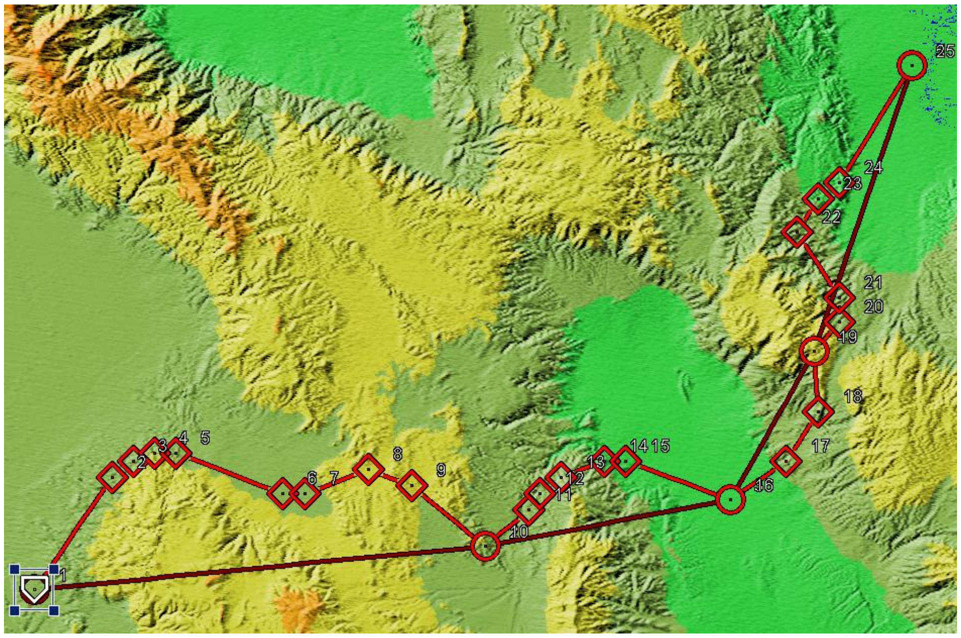


Figure 6.4 TFO Optimized Route: Nellis Route 1.

Several inferences can be made from inspection of Figure 6.4. First, TFO's approach to minimizing climbing involves seeking the lowest point possible. Inspection of the path between points 1 and 7 show that higher terrain was avoided whenever possible. This contrasts with the parallel path planner's approach which focused on minimizing the total amount of climbing. While TFO would avoid high terrain at any cost, the parallel path planner allows for high terrain so long as the cost of moving into the terrain is offset by reduced climbing and descending within the terrain. Table 6.2 compares the fitness evaluation of the TFO solution with the low cost and low risk Pareto front points of the parallel path planner.

Table 6.2 Nellis Route Evaluations.

Path	Path Length(m)	Climb Cost	Hideability
Low Cost	94289	8655	15696
Low Risk	93887	10272	14671
TFO	110931	11460	21758

Table 6.2 shows that both solutions of the parallel path planner had lower risk routes with shorter path lengths and lower climb costs. While the two programs have a different approach to minimizing climbing, it should be noted that the hideability algorithm and its input data are identical in both programs. Figure 6.4 also reveals a weakness in TFO's application of the restriction on heading change. Recall that consecutive target inputs in TFO must not result in a change in heading greater than 45 degrees. In the problem tested, this constraint was met. However, as each segment was optimized independently, the resulting solution contains a heading change greater than 90 degrees as seen around waypoint 10.

For completeness, the Pareto front for the same problem instance run as an MOEA is given in Figure 6.5.

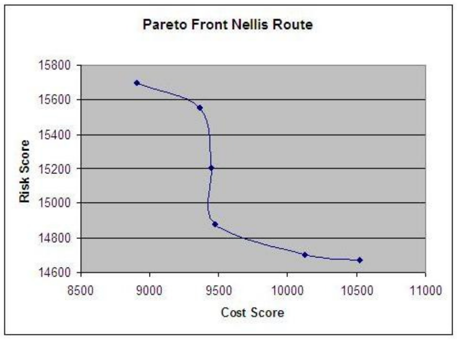
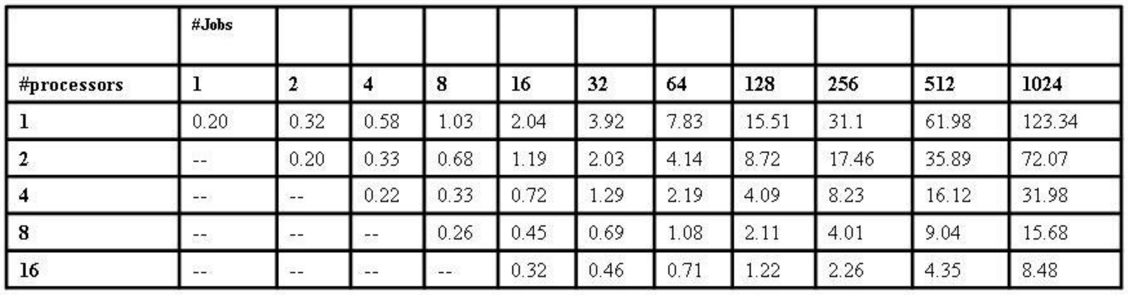


Figure 6.5 Pareto Front of Nellis Route Problem Instance.

6.1.4 Efficiency of the Parallel Path Planner. The experiments described in Section 5.1.4 reveal near linear speed-up. This is due to the independence of the nodes, and low communications overhead. Table 6.3 shows the runtimes in seconds for the various configurations tested.

Table 6.3 Average Runtimes for Various Parallel Configurations.



	#Jobs										
#processors	**1**	**2**	**4**	**8**	**16**	**32**	**64**	**128**	**256**	**512**	**1024**
1	0.20	0.32	0.58	1.03	2.04	3.92	7.83	15.51	31.1	61.98	123.34
2	--	0.20	0.33	0.68	1.19	2.03	4.14	8.72	17.46	35.89	72.07
4	--	--	0.22	0.33	0.72	1.29	2.19	4.09	8.23	16.12	31.98
8	--	--	--	0.26	0.45	0.69	1.08	2.11	4.01	9.04	15.68
16	--	--	--	--	0.32	0.46	0.71	1.22	2.26	4.35	8.48

From Table 6.3 it is clear that the parallelization of the path planner results in near linear speedup with each increase in the number of processors. This result is not unexpected as the parallel decomposition strategy has very low overhead. It should be noted however, that the load balancing scheme and the use of multiple non-blocking receives as discussed in Section 4.6, contributed to the speedup. In the absence of effective load balancing and non-blocking communication, the speedup would be reduced even with low-overhead parallel problem decomposition.

6.2 Evaluating the Improved Swarm Simulator.

6.2.1 Swarm Rules Experiments. This section presents results of swarm rules experiments 1-5 as defined in Section 5.3.2. The experiment was conducted from a single representative scenario. In each experiment, a large swarm (*n=100*) is broken into 5 sub swarms. The largest of these sub-swarms has a size *n=60*. The remaining sub-swarms each have a size *n=10*. The simulation is run for 300 seconds, during which time, each swarm executes a series of turn maneuvers.

- *Experiment 1: Path Following only*. The addition of path following behavior had a significant impact on swarm cohesion. Figures 6.6 and 6.7 present the results of experiment 1.

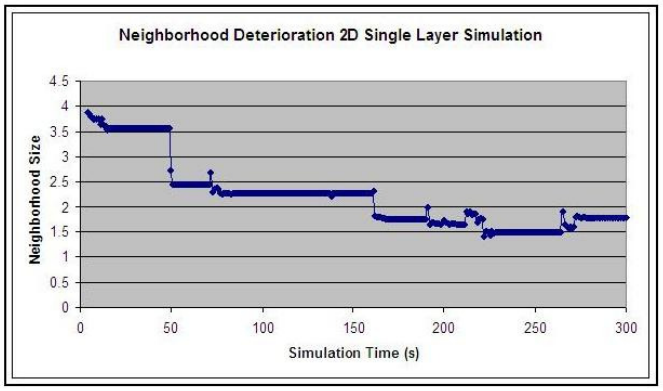


Figure 6.6 Experiment 1: Average Neighborhood Size.

At time t=0, the average neighborhood size is 3.8 which is consistent with the model. During the first 50 seconds, some minor deterioration is observed. Significant deterioration is observed after 50 seconds. Periodic spikes in the waveform reflect the effect of synchronization on neighborhood repair.

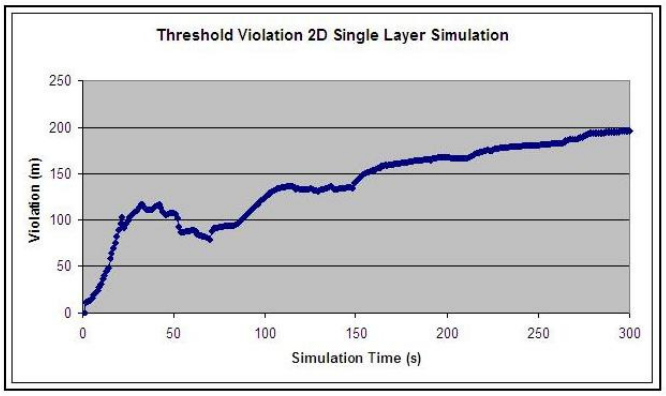


Figure 6.7 Experiment 1: Average Threshold Violation.

Figure 6.7 agrees with Figure 6.6. Rules violations increasing in magnitude over time indicate deterioration of the swarm. The effects of synchronization are seen more clearly during the first half of the simulation than in the second. This suggests that synchronization has diminishing returns with increased swarm fan out.

These effects were far less prominent with the four smaller sub swarms than they were on the large swarm. Data in figures 6.6 and 6.7 are taken from all swarm members. Figure 6.8 shows the threshold violations for the 50 swarm members assigned to the 4 small sub swarms.

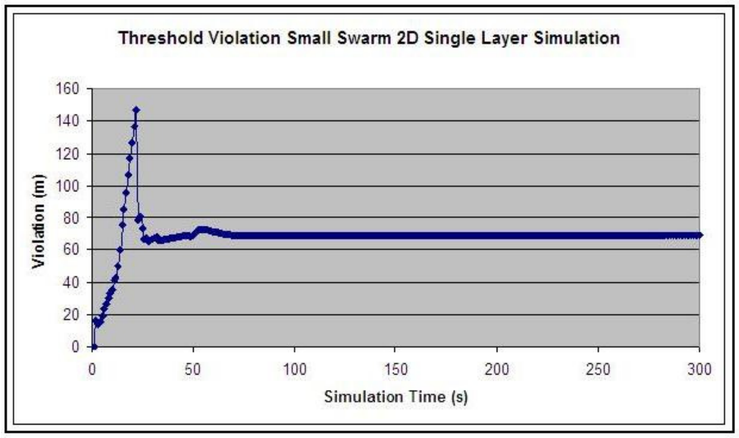


Figure 6.8 Experiment 1: Threshold Violations for Small Sun-swarms.

The initial set of violations between t=1 and t=25 are misleading. During this time, the swarm is performing a split operation. Individual members continue to observe their previous neighbors as they pull away in pursuit of their targets. Once the sub-swarms move away from one another the violations decrease. It is also clearly seen that the small sub-swarms maintain consistent behavior throughout the duration of the simulation.

- *Experiment 2: Single Vertical Layer Swarm with Terrain Following*. This experiment revealed additional stress on the model's ability to maintain swarm cohesion.

Once again, small sub-swarms were less affected by this enhancement. Figures 6.9 and 6.10 show the neighborhood deterioration and threshold violations experienced by the model with the addition of 3D terrain and terrain following behavior.

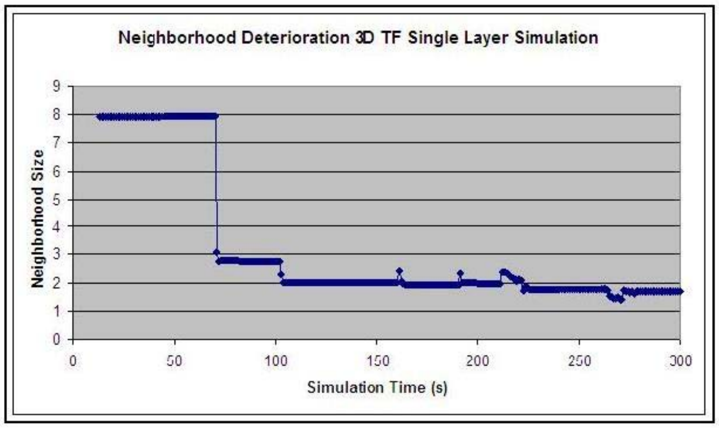


Figure 6.9 Experiment 2: Neighborhood Deterioration.

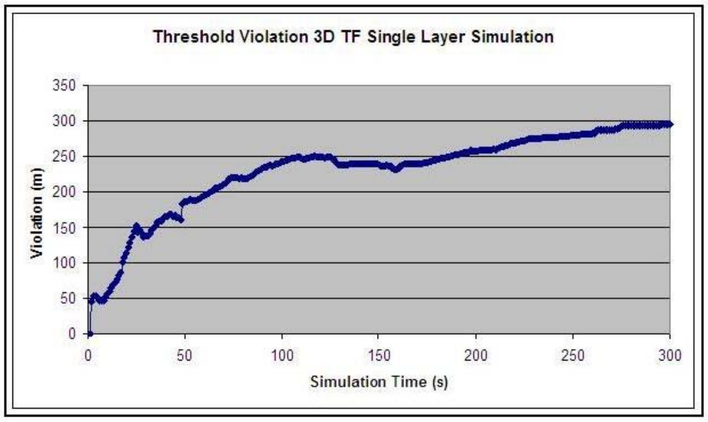


Figure 6.10 Experiment 2: Threshold Violation.

Figure 6.10 reveals that the model achieved the same instability after 50 seconds that the previous experiment reached by the end of the simulation. Figure 6.11 shows that rules violation increases with the addition of terrain following behavior for small swarm sizes. However, stability in the degree of violations is maintained after the split operation is performed.

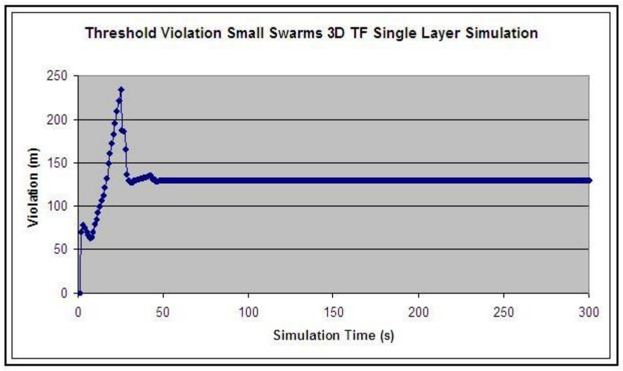


Figure 6.11 Experiment 2: Threshold Violation for Small Sub-swarms.

Figure 6.12 shows the extent of swarm fan out over time for this experiment.

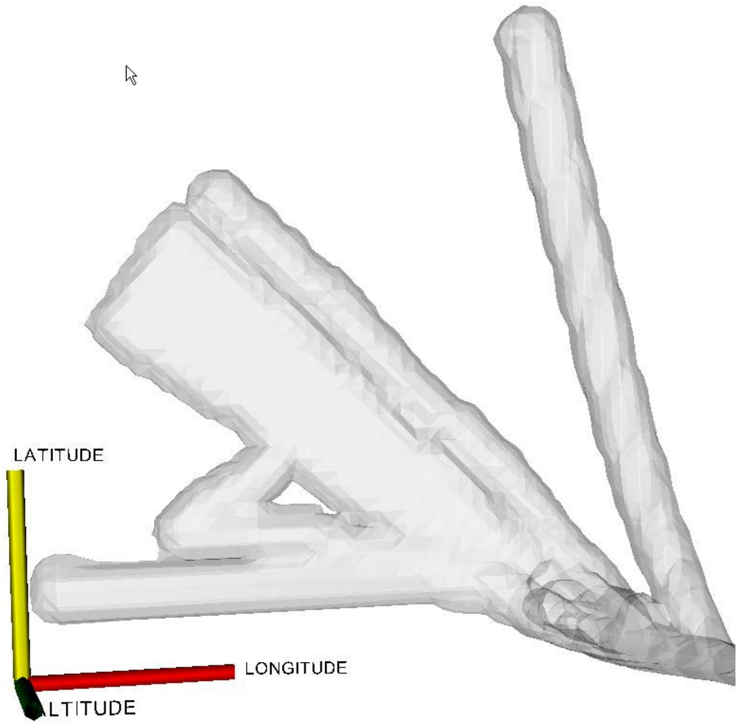


Figure 6.12 Experiment 2: Swarm Fan Out.

In Figure 6.12, the four smaller swarms appear left and right of the large swarm. It can be seen that each of the small swarms maintain cohesive while the larger swarm experiences greater fan out. Although fan out is obvious in the large swarm, it can be seen that the swarm still moves toward its destination, i.e. path following is maintained.

- *Experiment 3: Multi-Layer Swarms with Terrain Following.* In this experiment, the swarm consists if three vertical layers. The addition of multiple layers increases the number of neighborhood connections. Figures 6.13-6.15 present the results of experiment 3.

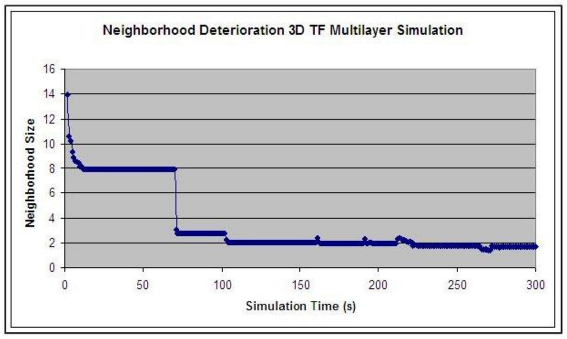


Figure 6.13 Experiment 3: Neighborhood Deterioration.

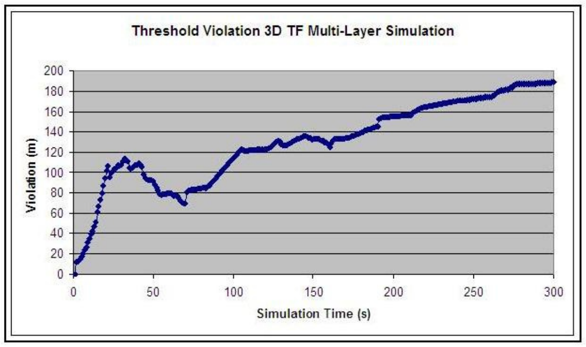


Figure 6.14 Experiment 3: Threshold Violations.

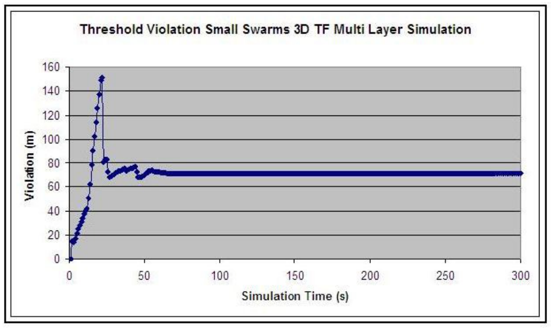


Figure 6.15 Experiment 3: Threshold Violations for Small Sub-swarms.

Comparison of the three graphs of experiment 3 with the graphs of experiment 2 show a significant increase in swarm stability with the addition of vertical swarm layers. The degree of rules violation was about half of what it was in the single-layer experiment. Furthermore, the effects of synchronization on swarm stability are more visible with the layered model. This is most obviously due to the increase in the number of neighborhood connections which drive the swarm behavior.

- *Experiment 4: Multi-Layer Swarms in 2D Environment.* In this experiment, the terrain following capability is removed from the model. While there is no noticeable change in the average neighborhood size between experiments 3 and 4, there is an increase in the degree of threshold violation. This phenomenon is curious given that in the 2-D multi-layered model, all vertical movement is aimed at maintaining swarm rules. In the 3-D model, vertical motion is generated with 80% emphasis on reaching the target and the remaining 20% focused on maintaining cohesion. Figures 6.16-6.18 show the neighborhood deterioration and threshold violation from experiment 4.

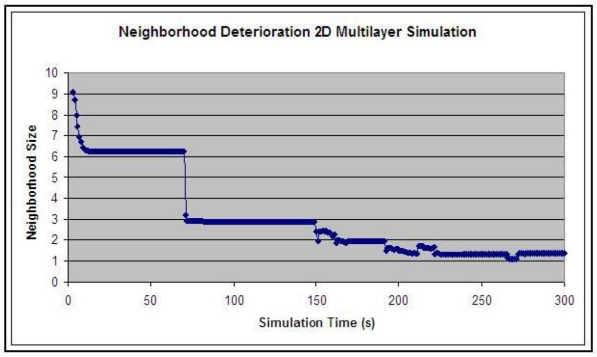


Figure 6.16 Experiment 4: Neighborhood Deterioration.

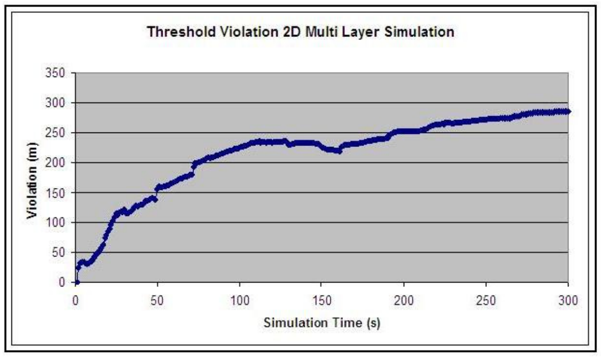


Figure 6.17 Experiment 4: Threshold Violation.

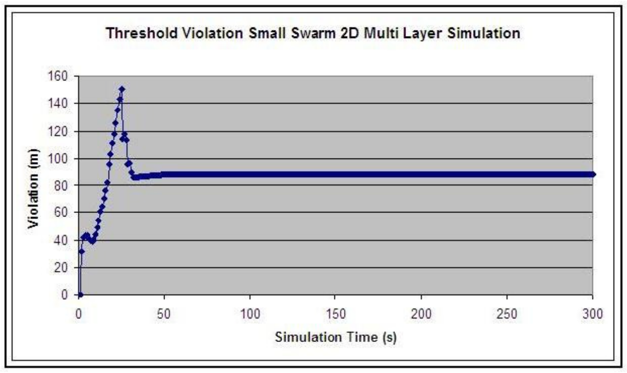


Figure 6.18 Experiment 4: Threshold Violation for Small Sub-swarms.

Experiment 5: The Effect of Swarm Synchronization on Swarm Stability. This experiment repeats the parameters of experiment 3, i.e. a multi-layered swarm with 3D terrain following. In addition, the period at which the swarm synchronizes is varied from 30 seconds (the baseline) to 90 seconds. Increased synchronization has an associated cost in terms of both computation and communication. Figure 6.19 shows the threshold violations for synchronization periods of 30, 60, and 90 seconds.

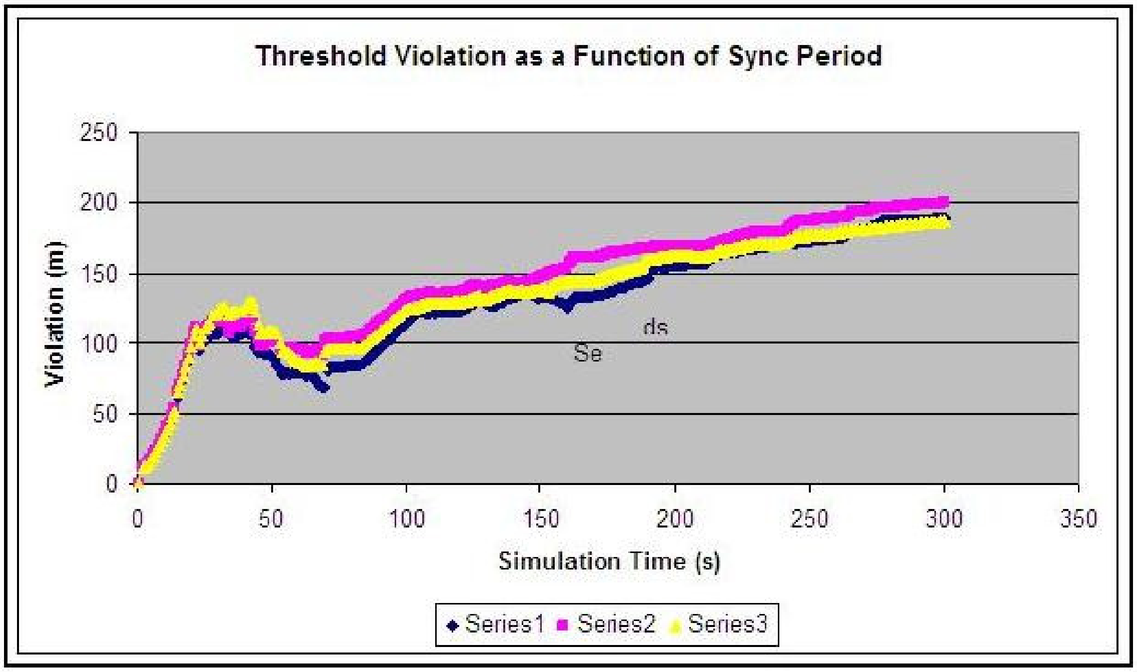


Figure 6.19 Experiment 5: The Effects of Synchronization Period on Model Stability.

Figure 6.19 shows that little benefit is seen in increased synchronization. There are two possible explanations for this: 1) after 60 seconds of simulation time, the swarm is too unstable to benefit from synchronization, and 2) the model forces synchronization during turn operations. The scenario contains numerous turns in the swarm's path. Additional scheduled synchronization may realize little additional benefit if forced synchronization occurs regularly.

6.2.2. Simulation Scalability Experiment. This experiment showed that the model has limited scalability. Parallelization of the simulation however, provides nearly linear speedup for larger problem sizes. Table 6.4 and Figure 6.20 show the runtimes in seconds for a 20 minute sample of the simulation.

Table 6.4 Runtimes for Parallel Problem Execution.

	UAVs			
CPUs	**40**	**80**	**160**	**320**
4	43	174	1873	36657
8	21	97	949	18330
16	22	64	496	9278

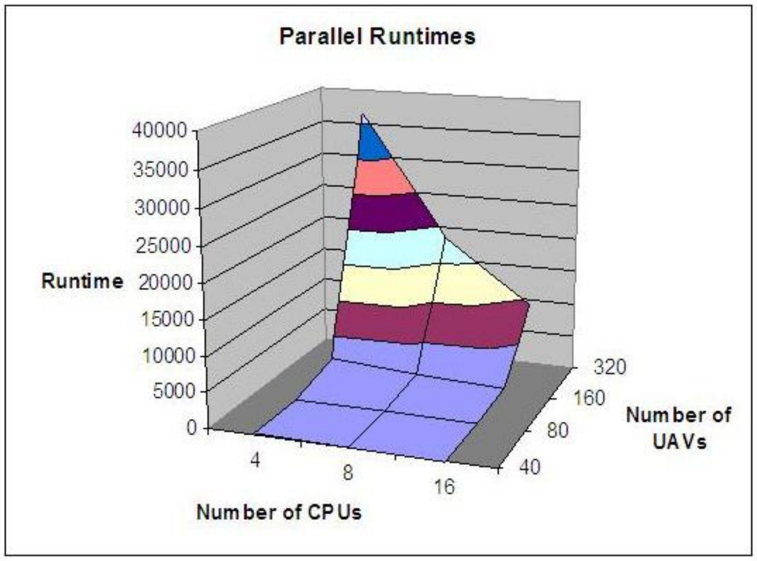


Figure 6.20 Parallel Runtimes as a Function of Problem Size and Number of CPUs.

Table 6.4 and Figure 6.20 suggest limitations in the scalability of the model. A significant increase in runtime is observed as the problem size increases from 160 to 320 UAVs. Speedup resulting from the additional processors is however nearly linear as a column-wise inspection of table 6.4 suggests. The poor scalability may result from the changes in neighborhood resulting from the model's instability. As neighborhoods are reconfigured, the model requires direct communication among swarm members to adjust their positions. Chapter 7 presents some alternative communication methods from the literature.

6.3. Summary. This chapter presents the results of the complete experiment set defined in chapter 5. Analysis of the data reveals opportunities for improvement of the parallel path planner and the swarm behavior model. Chapter 7 provides conclusions on the effectiveness of the AFIT UAV Swarm Mission Planning and Parallel Simulation System and suggestion to improve the performance of the individual components.

7. Conclusion and Future Research

The overall goals of this research are met. The AFIT UAV Swarm Mission Planning and Parallel Simulation System integrates three domains that have, in past efforts, been treated independently. The system architecture is capable of integrating additional functionality as it becomes available. This chapter addresses the satisfaction of the research objectives as defined in Chapter 1 and provides recommendations for future research.

7.1 Satisfaction of Objectives. Recalling the Chapter 1 objectives:

1. Develop a multi-objective evolutionary algorithm for efficient path planning
2. Develop an efficient parallel system that computes individual segments for use in the GVR routing algorithm
3. Improve AFIT's parallel swarm simulator by incorporating path-following capabilities with existing swarm behavior and measure the effects of these capabilities on swarm cohesiveness.

The results presented in Chapter 6 show that objectives 1 and 2 are satisfied. For example, the path planner generates paths that avoid terrain as seen in figure 6.1. The set of solutions to the multi-objective routing problem defined in Section 5.1 produces a set of solutions with cost vs. risk tradeoffs. These solutions have lower cost and associated risk than those produced by the Terrain Following Optimizer as seen in table 6.2. The efficiency of the path planner's parallelization scheme is clearly demonstrated in table 6.3.

Objective 3 was satisfied in terms of the addition of path following behavior. The data in Section 6.2 show that the addition of path following and terrain following behaviors to the swarm model produces stable behavior for small swarm sizes (<12 UAVs). Instability was observed in the behavior of larger swarms (>50 UAVs) as a result of the path following behavior. Additional research is needed to mitigate the side effects of these additional capabilities as they pertain to swarm cohesion in large swarm formations. Possible insight was gained into the limitations of fixed-wing flight as an option for creating physically realizable swarms. For example, slower cruise speeds and greater maneuverability may be needed to implement some of the lateral movement proposed by the swarm model.

7.2 Improvements to the Path Planner. Compared to the Terrain Following Optimizer (TFO), the path planner generates routes that are comparable. Additionally, the turning violations generated by TFO and the restrictions on the spacing of targets were eliminated. Future improvements to the path planner are needed regarding its ability to explore larger-sized areas for terrain and threat avoidance. Currently, the search area for the placement of points is tightly restricted by the turn constraint. This restriction causes the planner to behave more like a local search algorithm than a true optimization tool. More exploratory mutation operators such as Xiao's *Mutate 2* [67] that deletes multiple consecutive segments and replaces them with new ones could overcome this deficiency.

Another promising technique is to increase the search space through the use of "migrant" population members. Originally developed for use in the Island model [61], migrant members are randomly initialized solutions added to the population at various epochs of the evolutionary cycle. The diversity added to the population would allow the planner to search different regions of the problem space.

As a multi-objective algorithm, the path planner creates an aggregation of the fitness characteristics into two fitness functions. The elements of each function are parameterized. For example, the cost function combines the path length and climb cost with a parameterized weighting factor for each. These parameters have been derived empirically in the development of TFO and they have not been well documented. If the benefit of creating an MOEA is to provide the decision maker with trade-offs, then further research into the parameterization of the cost and risk elements would enhance this knowledge. The parameters could be input directly by decision makers if a heuristic were available to guide the decision.

In their current form, both TFO and the parallel path planner have no conception of "time on target" in their approach to optimization. Mission planning generally requires targets be reached by specified times. Modifications to the path planner should allow either verification that time on target constraints can be met or that adjustments in the vehicle speed be evolved along path segments.

Yet another weakness of the TFO and parallel path planner is the lack of a parameterized vehicle model. Both systems are tuned to create path feasibility based on a liberal first-order approximation of the AC-130 gunship. The internal constraints on path feasibility need to be fully parameterized so that any vehicle model can be used in conjunction with the path planner. Clearly, more maneuverable vehicles would be subject to softer constraints, thus enabling greater levels of optimization in the path.

7.3 Improvements to the Parallel Swarm Simulator. In transitioning the swarm behavior model, three major capabilities were added: Path following, Terrain following, and 3-D swarm formation. The results described in chapter 6 show that while path following allowed vehicles to reach their destinations, deterioration of swarm cohesiveness was observed, especially for larger swarms. There are several explanations for this. In the previous swarm model, the distribution of weights for cohesion and alignment vectors was nearly 50-50. In the current model, Target vectors were given 80% of the total vector weight while the total weight of all neighborhood-driven cohesion vectors was only 20%. This limitation is placed in the model to serve as a place-holding constraint. Large vehicles traveling at high speeds do not have the ability to create rapid lateral movement. Therefore there are general physical limitations on the degree to which cohesion can be achieved while still exceeding the vehicle's stall speed. The place-holding constraint can be replaced by a realistic 2nd order flight model. With such a model, greater lateral movement to achieve cohesion could be achieved providing the vehicle had lower stall speeds. A second hindrance to maintaining cohesion is the model's limitation on the amount of increase or decrease on the thrust during a turn maneuver. In the current model, a vehicle can only speed up or slow down by 10% during a simulation time step. Again this is a place-holding constraint which takes into account in a general way the limitations on a vehicle's ability to accelerate. A second order vehicle model would provide realistic bounds on acceleration that would juxtapose required behavior against realistic capability.

The greater deterioration of cohesion in large swarms versus small swarms suggests possible weaknesses in the nearest neighbor model. While individual neighborhoods are fairly well maintained, the neighborhood of neighborhoods within the large swarms clearly deteriorates. Current research in control systems provides some additional possibilities to address this concern. Bacconi proposes a Command Governor approach to maintain formations of satellites [1]. In his model, a command entity directs the movements of individuals. The individual satellites use the commander's position as a reference frame and adjust their position relative to the commander. In a swarm of vehicles, a tiered command approach could be used to manage swarm movement. Each neighborhood could have a central figure or "block captain" that directs local movement. This individual would then communicate to the next level of command above which would be a central figure responsible for directing the movement of a group of neighborhoods. Using this tree-based hierarchy, the number of levels of communication would need to grow as the log of the swarm size.

7.4 Future Simulation Capabilities. In addition to improvements required in the current swarm model, additional capabilities would enhance the simulation. The mission planning scenario in this research assumes a priori knowledge of all mission objectives and a fixed resource capability. While it is reasonable to assume that missions are created to complete known objectives, additional information gained throughout the mission could alter the objectives. Some swarm behavior models [42] [51] have a searching behavior defined that allows the swarm to seek out unknown targets and adjust behavior if and when they are detected. While Price's model is 2-D and does not include any path following capability, the self-organization and discovery capabilities would enhance the AFIT swarm simulator if they could be integrated with existing capabilities.

The result of such integration would be a swarm model that can perform a defined strike or reconnaissance mission while adapting behaviors to account for enemy engagement, attrition, and the discovery of targets of opportunity.

Appendix A. Modified Convex Hull Algorithm

This appendix describes the development of the Modified Convex Hull Algorithm developed by J. Zaloudek. The algorithm is used in the repair of paths to create additional way points such that no turn exceeds 45 degrees.

Step 1. Place the octagon such that the midpoint, P_m is one vertex in it, chosen to be as close to the middle of the turn as possible.

Step 2. Find the convex hull (is like the "gift wrapping").

Step 3. Traverse the CH route from P_o to P_f (this will require going through the CH route backwards if the point immediately after P_o in the CH route is Pf).

Placement of the octagon. Optimally the octagon could be placed such that Pm is half-way through the turn. The line (P_s-P_m) represents a normal to the turn at point Pm if Pm were exactly halfway through the turn. Due to the grid, it will most likely not be possible to do this perfectly, but the angle of the line P_s Pm can be used to place the octagon as close as possible. Figures A.1 and A.2 Illustrate the Octagon Placement

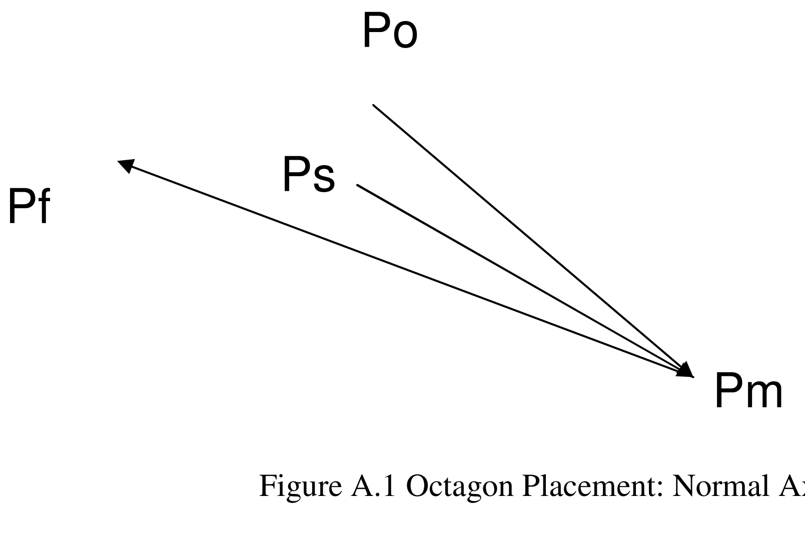


Figure A.1 Octagon Placement: Normal Axis

Ps-Pm angle (with respect to the x axis)	Vert to place on Pm
< π/4	0
< π/2	1
< 3π/4	2
< π	3
< 5π/4	4
< 3π/2	5
< 7π/4	6
< 2π	7

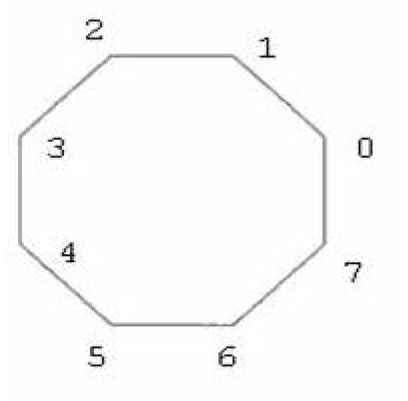


Figure A.2 Placement of Octagonal Vertices on Midpoint

To reference the octagon, two arrays of coordinate offsets are used.

```
int xOctOffsetes = [0,-1,-2,-3,-3,-2,-1, 0];

int yOctOffsetes = [0, 1, 1, 0,-1,-2,-2,-1];
```

As an example, suppose P_m is (3,4) and the angle of P_s-P_m is 7π/4. Vertex 7 of the octagon would be chosen as the one to go on P_m. Back out the location of vertex 0 by subtracting the offsets for vertex 7 from P_m's coordinates: vert0 then is (3,5). Going through the list and adding the offsets to vert0's coordinates then specifies all the points in the octagon.

Appendix B. Basic MPI Constructs

This appendix describes the basic MPI constructs used in the parallel path planner.

MPI_Init. All MPI programs begin with a call to *MPI_Init* [Pacheco][Snir]. This function performs set-up operations that allow the MPI library routines to be used. No MPI commands can precede this call.

MPI_Finalize. After the last MPI command is used, *MPI_Finalize* is called to clean up the system and de-allocate any memory used. No MPI functions may follow the call to *finalize*.

MPI_Comm_rank. Most MPI programs have a master/slave organization. *MPI_Comm_rank* is a function that looks at all nodes in the program through a variable called *MPI_COMM_WORLD* and determines the rank or node number of the individual program instance. An integer reference variable, &me, is passed to the function and is set with the rank of the individual. By convention, the node with rank 0 is used as the master, while all others are used as worker nodes. The rank variable is used in the decision tree to determine whether the program instance follows the code associated with the master node or that of a working node.

MPI_Comm_size. Since each node is assigned a number, the function *MPI_Comm_size* returns the number of nodes in the program. The size or number of nodes is often used as a check to ensure that all nodes have completed their tasks.

Blocking and non-blocking communication. MPI allows for both blocking and non-blocking communication. Blocking communication halts the sending program instance until the corresponding receive has completed. In non-blocking communication, the calling program continues to execute without waiting. In this program, both blocking and non-blocking communication are used.

MPI_Bcast. To send common data such as terrain elevations to all nodes, the broadcast method is called by the master node. The parameters of this function include buffer to be sent, the size, the data type, the sender, and the variable *MPI_COMM_WORLD*. Broadcasts are non-blocking. In order for the master to use the broadcast, the slave nodes must also call *MPI_Bcast*. Both the broadcast sender and the receiver use the node of the caller as a parameter. If the name of the node in the function call matches that of the calling node, the system interprets this as a *send*. If calling node is different from the node in the function call, the system interprets this as a *receive*.

MPI_Barrier. In a parallel program it is often necessary to wait for all nodes to reach a common point before proceeding. When a node reaches an *MPI_Barrier* call, it halts until all other nodes in MPI_COMM_WORLD reach the same point.

MPI_Send. This blocking function is the most primitive of the MPI send functions. It is a point-to-point message that sends a buffer of data having a declared size and type to a designated receiver.

MPI_Recv. Also a blocking communication, this function halts the calling program until the specified buffer receives the message from the designated sender. Use of this function is appropriate in cases where the calling program needs to process data in the buffer as its next task.

MPI_Irecv. When a program is waiting on multiple nodes to transmit, data, this non-blocking receive can be used. It allows the receiving program to receive messages in the order they come in. Without this non-blocking ability, the calling program would have to block and receive each message in turn.

MPI_Request. When non-blocking receives are used, this accessory data structure provides a mechanism to check for a change in receipt status. It is used with a *MPI_Status* flag that is set to true when the corresponding *MPI_Request* has been met.

MPI_Test. This function uses both the *MPI_Request* and *MPI_Status* flag associated with the non-blocking receive. *MPI_Test* is called and if the status flag associated with the *MPI_Request* is set to true, then the receive has been completes and the calling program can begin processing the data in the receive buffer.

Bibliography

1. Bacconi, F. and Mosca, E. "A Command Governor Approach to Formation Flying Control Problems." *Proc of 16th IFAC Symposium on Automation and Control* St. Petersburg, Russia: 1700-1730 (2004).

2. Back, T. *Evolutionary Computation 1: Basic Algorithms and Operators.* Philadelphia: Institute of Physics Publishing, (2000).

3. Barraquand, J. and Ferbach, P. "Path Planning Through Variational Dynamic Programming" Paris Research Laboratory, Paris (1993) 13 November 2005 ftp.digital.com/pub/Digital/ PRL/research-reports/PRL-RR-33.pdf

4. Batavia, P. "Path Planning for the Cye Personal Robot." Carnegie Mellon University Robotics Institute, PA (2002).

5. Brown, Darrin. *Routing Unmanned Aerial Vehicles While Considering General Restricted Operating Zones.* MS Thesis, AFIT/GOR/ENS/01M-04. Graduate School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH March 2001.

6. Canny, J. *The Complexity of Robot Motion Planning*. MIT Press, New York (1988).

7. Carriker, W. "Path Planning for Mobile Manipulators for Multiple Task Execution." *Proceedings of the 1991 IEEE International Conference on Robotics and Automation* 7(3):403-408 (1991).

8. Carriker, W. "The Use of Simulated Annealing to Solve the Mobile Manipulator Path Planning Problem," *Proceedings of the 1990 IEEE International Conference on Robotics and Automation* (1):204-209 (1990).

9. Castillo, Oscar and Trujillo, J. "Multiple Objective Optimization Genetic Algorithms for Path Planning in Autonomous Mobile Robots," *International Journal of Computers, Systems, and Signals,* 6 (1):48-62 (2005). 14 November 2005

10. Cobb, H. "An Investigation into the use of Hypermutation as an Adaptive Operator in Genetic Algorithms Having Continuous, Time-dependent Non-stationary Environments," Technical Report AIC-90-001, Naval Research Laboratory, Washington (1990).

11. Coello, C and Van Veldhuizen, David. "Evolutionary Algorithms for Solving Multi-Objective Problems." Kluwer Academic Publishers, Norwell, MA 2002.

12. Cormen, T et al. Introduction to Algorithms. New Delhi: Prentice-Hall of India (2004).

13. Corner, Joshua and Lamont, Gary “Parallel Simulation of UAV Swarm Scenarios” *Proc. of the 2004 Winter Simulation Conference*. 355-363 (2004)

14. Corner, Joshua. *Swarming Reconnaissance Using Unmanned Aerial Vehicles in a Parallel Discrete Event Simulation.* MS Thesis, AFIT/GCE/ENG/04-01. Graduate School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH March 2004.

15. Correa, E. “A Genetic Algorithm for the P-median Problem,” *Proceedings of The Genetic and Evolutionary Computations Conference* pp 1268-1275. San Francisco (July 2001). 15 November 2005

16. Damitio, M. “Comparing the Breathing Time Buckets Algorithm and the Time Warp Operating System on a Transputer Architecture,” *In Proceedings of the SCS European Simulation Multi-Conference.* 141-145 (1994)

17. Davis, P. “B-Splines and Geometric Design,” *Slam News* 29(5) June 1996. 17 November 2005 http://users.wpi.edu/~pwdavis/sinews/spline/spline17.htm

18. Deb, K. “Multi-Objective Optimization Using Evolutionary Algorithms.” Hoboken, NJ: Wiley. (2001)

19. ---- “F11-Aardvark.” Air Force Link webpage. 20 April 2006: http://www.af.mil/history/aircraft.asp?dec=&pid=123006568

20. ---- SkyView. *Georgia Tech Research Institute Homepage*. 5 Jan 2005 http://www.gtri.gatech.edu/ittl/csit/proj_skyview.html

21. M. Garey. “Computers and Intractability, A Guide to Theory of NP-Completeness.” W.H Freeman, New York. 1979

22. Jia, Dong. “Parallel Evolutionary Algorithms for UAV Path Planning.” *Proceedings of the AIAA 1st Intelligent Systems Conference*. (2004)

23. Kadrovach, A. *A Communication Modeling System for Swarm-based Sensor Networks.* PhD dissertation, Graduate School of Engineering and Management, Air Force Institute of Technology, Wright-Patterson AFB, OH (2003).

24. Kleeman, Mark and Lamont, Gary. “Solving the Aircraft Engine Maintenance Problem Using a Multi-objective Evolutionary Algorithm.” *Proceedings of the 3rd Intl. Conf. Evolutionary, Multi-Criterion Optimization, EMO 2005* p782 (2005).

25. Kuffner, J. “RRT_Connect: An efficient Approach to Single-Query Path Planning,” Computer Science Department, Stanford University, Stanford CA (2002).

26. Ladd, A and Kavraki, Lydia. “Fast Tree-Based Exploration of State Space for Robots with Dynamics.” Department of Computer Science, Rice University, Huston, TX (2004).

27. Lancet, M. “GALib Download Page” *Leaders for Manufacturing*, Massachusetts Institute of Technology (2000). 12 September 2005 http://lancet.mit.edu/ga

28. LaValle, Stephen. “Planning Algorithms” ch5, pp 193-196 Cambridge University Press, MA. 2006

29. LaValle, Stephen. “Rapidly-Exploring Random Trees: A New Tool for Path Planning,” Department of Computer Science, Iowa State University, Ames. IA. (1998).

30. Lozano, P. “A Simple Motion-Planning Algorithm for General Robot Manipulators,” *IEEE Transactions on Robotics and Automation*, 3(3), 224-238 (1987)

31. Lua, C. “Synchronized Multi-point Attack by Autonomous Reactive Vehicles with Local Communication,” *Proceedings of the 2003 IEEE Swarm Intelligence Symposium.* (Dec 2002).

32. Mataric, M. “Issues and Approached in the Design of Collective Autonomous Agents,” *Robotics Automations Systems,* 16(2-4):321-331 (1995).

33. Matsumoto, M. "Dynamic Creation of Pseudorandom Number Generators", Monte Carlo and Quasi-Monte Carlo Methods, 56-69 Berlin: Springer (2000)

34. Mazer, E. “The Ariadne’s Clew Algorithm,” *Journal of Artificial Intelligence Research,* 9 295-316 (1998). 11 November 2003 http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume9/mazer98a-html/ariane.html

35. Mezouar, Y. “Path Planning for Robust Image Control.” *IEEE Transactions on Robotics and Automation.* 18:4 2002

36. Milam, K. *Evolution of Control Programs for a Swarm of Autonomous Unmanned Aerial Vehicles.* MS Thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH. (March 2004).

37. Mittal, S and Deb, K. "Three-Dimensional Offline Path Planning for UAVs Using Multiobjective Evolutionary Algorithms," Kanpur Genetic Algorithms Laboratory Report: 20040008, Indian Institute of Technology, Kanpur, India (2004).

38. Michaelwicz, Z and Fogel, D. *How to Solve It: Modern Heuristics.* Berlin: Springer-Verlag (2000).

39. Nikolos, I. "Evolutionary Algorithm Based Offline/Online Path Planner for UAV Navigation," *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics* 33 (6):898-912 (Dec 2003).

40. Pacheco, P. *Parallel Programming with MPI.* San Francisco: Morgan Kauffman 1997.

41. Plaku, E. et al. "Sampling-Based Roadmap of Trees for Parallel Motion Planning." Department of Computer Science, Rice University, Houston, TX (2004).

42. Price, Ian. *Evolving Self Organizing Behavior for Homogeneous and Heterogeneous Swarms of UAVs and UCAVs.* MS Thesis, AFIT/GCS/ENG/06M-11. Graduate School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH March 2006.

43. Parunak, H. "Swarming Coordination of Multiple UAVs for Collaborative Sensing." *Proc of 2nd AIAA Unmanned Unlimited Systems Technologies and Operations Aerospace Land, and Sea Conference.* San Diego, CA (2003).

44. Rathbun, David. *Evolutionary Approaches to Path Planning Through Uncertain Environments.* AIAA UAV Conference, Portsmouth, VA #2002-3455 (2002).

45. Reynolds, Craig. "Flocks, Herds, and Schools: A Distributed Behavior Model," *Computer Graphics*, 21(4):25-34 (1987).

46. Reynolds, Craig. "Steering Behaviors for Autonomous Characters," *Proceedings of The Game Developers Conference.* San Jose, California. 763-782. (1999).

47. Reynolds, P.F. "A Spectrum of Options for Parallel Simulation," *Proc. of Winter Simulation Conference.* 325-332 (1988)

48. Russell, Mathew. "On Using SPEEDES as a Platform for UAV Swarm Simulation." *Proceedings of the 2005 Winter Simulation Conference* 1130-1137 (2005).

49. Russell, Mathew. *A Genetic Algorithm for the UAV Routing Problem Integrated with a Parallel Swarm Simulation*. MS Thesis, AFIT/GCS/ENG/05M-16.

Graduate School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH March 2005.

50. Sanchez, G. "Locally-Optimal Path Planning by Using Probabilistic Roadmaps and Simulated Annealing." *Proceedings of IASTED International Conference on Robotics and Applications (RA-99)*, October 28-30, Santa Barbara. (1999).

51. Scheutz, M. "The Utility of Heterogeneous Swarms of Simple UAVs with Limited Sensory Capacity in Deduction and Tracking Tasks." *In Proc. of the 2005 IEEE Swarm Intelligence Symposium.* Pasadena, CA (2005).

52. Scott, A. "B-Splines" *The Connections Project.* (Apr 2003) 16 November 2005 http://cnx.rice.edu/content/m11129/latest

53. Sezer, Ergin. *Mission Route Planning with Multiple Aircraft & Targets Using Parallel A* Algorithm.* MS Thesis. AFIT/GCS/ENG/01M-06. College of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, OH (2000)

54. Snir, M. *MPI – The Complete Reference, Volume 1, The MPI Core.* Cambridge: MIT Press 2000.

55. -----. "SOCOMS Terrain Following Radar Competition." Defense Industry Daily Website. 1 Mar 2006: http://www.af.mil/history/aircraft.asp?dec=&pid=123006568

56. -----. *SPEEDES API Reference Manual*. Solana Beach CA: Metron, Inc., January 2002 (DN: 2025 v2)

57. -----. *SPEEDES Users Guide*. Solana Beach CA: Metron, Inc., April 2003 (DN: 2024 v4)

58. Steinman, J. "Breathing Time Warp," *Workshop on Parallel and Distributed Simulation: Proceedings of the 7th Workshop on Parallel and Distributed Simulation.* 109-118 San Diego (1993)

59. Sugihara, K and Smith, J. "A Genetic Algorithm for 3-D Path Planning of a Mobile Robot." Department of Information and Computer Science, University of Hawaii, Manoa, Honolulu, (1999)

60. Tanese, R. "Parallel Genetic Algorithms for a Hypercube." *Proc of the 2nd Intl. Conf. on Genetic Algorithms* Hillsdale, NJ: 177-183 (1987)

61. Tavares, J. "GVR Delivers it on time". *4th Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'02),* 745-749. 2002.

62. Toth, Paolo and Vigo, D. *The Vehicle Routing Problem*. Philadelphia: Society for Industrial and Applied Mathematics (2002).

63. Trahan, M. et al. "Swarms of UAVs and Fighter Aircraft," *Proceedings of the 2nd Intl. Conf. on Non-Linear Problems in Aviation and Aerospace*. (2) 745-752. (1998).

64. Trevelyan, J. "Approximating B-Splines," Department of Mechanical Engineering, University of Western Australia, Crawley, Australia (2002) 1 April 2006 http://www.mech.uwa.edu.au/jpt/matlab-dxf

65. Warren, C. "Global Path Planning Using Artificial Potential Fields." *IEEE Transactions on Robotics and Automation Proceedings*, 316-381 1989.

66. Weisstein, E. et al. "Manifold," *Mathworld, a Wolfram Web Resource*. 13 November 2005 http://mathworld.wolfram.com/Manifold.html

67. Xiao, J. and Michalewicz, Z. "Adaptive Evolutionary Planner/Navigator for Mobile Robots," *IEEE Transactions on Evolutionary Computation* 1 (1):18-28 (April 1997)

68. Yavuz, Kurtsat. *Multi-Objective Mission Route Planning Using Particle Swarm Optimization.* MS Thesis AFIT/GCS/ENG/02M-12. College of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, OH (2002).

69. Zitzler, E. and Deb, K. "Comparison of Multi-objective Evolutionary Algorithms: Empirical Results." *Evolutionary Computation 8*(2):173-195 (2000).

REPORT DOCUMENTATION PAGE						*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* 13-06-2006	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From - To)* May 2005 - June 2006

4. TITLE AND SUBTITLE AFIT UAV Swarm Mission Planning and Simulation System	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Slear, James, N., Captain, USAF	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GCE/ENG/06-08
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFRL/IFSC Mr. Derryl Williams 2431 Avionics Circle Bldg 620 (AREA B) WPAFB OH 45433 (DSN)785-4827 — AFRL/SNZW Mr. Mike Foster 2431 Avionics Circle Bldg 620 (Area B) WPAFB OH 45433 (DSN)785-5900	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The purpose of this research is to design and implement a comprehensive mission planning system for swarms of autonomous aerial vehicles. The system integrates several problem domains including path planning, vehicle routing, and swarm behavior.
The developed system consists of a parallel, multi-objective evolutionary algorithm-based path planner, a genetic algorithm-based vehicle router, and a parallel UAV swarm simulator. Each of the system's three primary components are developed on AFIT's Beowulf parallel computer clusters.
Novel aspects of this research include: integrating terrain following technology into a swarm model as a means of detection avoidance, combining practical problems of path planning and routing into a comprehensive mission planning strategy, and the development of a swarm behavior model with path following capabilities.

15. SUBJECT TERMS
Multi-objective Evolutionary Algorithms, Autonomous Unmanned Aerial Vehicles, Swarms, Parallel Processing, Parallel Discrete Event Simulation

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Dr. Gary B. Lamont (ENG)
U	U	U	UU	151	**19b. TELEPHONE NUMBER** *(Include area code)* (937)-255-3636 x4718